

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of

Taketo HEISHI et al.

Serial No. NEW

Filed July 31, 2003

THE COMMISSIONER IS AUTHORIZED  
TO CHARGE ANY DEFICIENCY IN THE  
FEES FOR THIS PAPER TO DEPOSIT  
ACCOUNT NO. 23-0975

Attn: APPLICATION BRANCH

Attorney Docket No. 2003\_1076A

COMPILER, COMPILER APPARATUS  
AND COMPILATION METHOD

**CLAIM OF PRIORITY UNDER 35 USC 119**

Commissioner for Patents  
P.O. Box 1450  
Alexandria, VA 22313-1450

Sir:

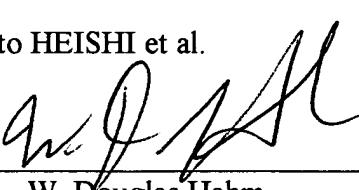
Applicants in the above-entitled application hereby claim the date of priority under the International Convention of Japanese Patent Application No. 2002-226682, filed August 2, 2002, as acknowledged in the Declaration of this application.

A certified copy of said Japanese Patent Application is submitted herewith.

Respectfully submitted,

Taketo HEISHI et al.

By

  
W. Douglas Hahm  
Registration No. 44,142 *for*  
Michael S. Huppert  
Registration No. 40,268  
Attorney for Applicants

WDH/kjf  
Washington, D.C. 20006-1021  
Telephone (202) 721-8200  
Facsimile (202) 721-8250  
July 31, 2003

日本特許庁  
JAPAN PATENT OFFICE

別紙添付の書類に記載されている事項は下記の出願書類に記載されて  
いる事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed  
with this Office

出願年月日

Date of Application: 2002年 8月 2日

出願番号

Application Number: 特願2002-226682

[ ST.10/C ]:

[ JP2002-226682 ]

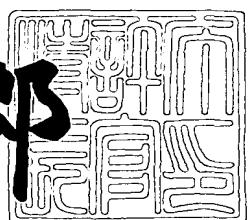
出願人

Applicant(s): 松下電器産業株式会社

2003年 2月 28日

特許庁長官  
Commissioner,  
Japan Patent Office

太田 信一郎



出証番号 出証特2003-3011925

【書類名】 特許願  
【整理番号】 5037740106  
【あて先】 特許庁長官殿  
【国際特許分類】 G06F 9/45  
【発明者】  
【住所又は居所】 大阪府門真市大字門真1006番地 松下電器産業株式  
会社内  
【氏名】 瓶子 岳人  
【発明者】  
【住所又は居所】 大阪府門真市大字門真1006番地 松下電器産業株式  
会社内  
【氏名】 坂田 俊幸  
【発明者】  
【住所又は居所】 大阪府門真市大字門真1006番地 松下電器産業株式  
会社内  
【氏名】 小川 一  
【発明者】  
【住所又は居所】 大阪府門真市大字門真1006番地 松下電器産業株式  
会社内  
【氏名】 宮地 涼子  
【発明者】  
【住所又は居所】 大阪府門真市大字門真1006番地 松下電器産業株式  
会社内  
【氏名】 宮阪 修二  
【発明者】  
【住所又は居所】 大阪府門真市大字門真1006番地 松下電器産業株式  
会社内  
【氏名】 石川 智一

【特許出願人】

【識別番号】 000005821

【氏名又は名称】 松下電器産業株式会社

【代理人】

【識別番号】 100109210

【弁理士】

【氏名又は名称】 新居 広守

【電話番号】 06-4806-7530

【手数料の表示】

【予納台帳番号】 049515

【納付金額】 21,000円

【提出物件の目録】

【物件名】 明細書 1

【物件名】 図面 1

【物件名】 要約書 1

【プルーフの要否】 要

【書類名】 明細書

【発明の名称】 コンパイラ用プログラム、コンパイラ装置及びコンパイル方法

【特許請求の範囲】

【請求項1】 ソースプログラムを機械語プログラムに翻訳するコンパイラ用のプログラムであって、

対象とするプロセッサに特有の機械語命令に対応する演算処理が予め定義された演算処理定義情報を含み、

前記プログラムは、

前記ソースプログラムを解析するパーサーステップと、

解析されたソースプログラムを中間コードに変換する中間コード変換ステップと、

変換された中間コードを最適化する最適化ステップと、

最適化された中間コードを機械語命令に変換するコード生成ステップとを含み

前記中間コード変換ステップは、

前記中間コードのうち、前記演算処理定義情報を定義された演算処理を参照しているものがあるか否かを検出する検出サブステップと、

あると検出された場合に、その中間コードを、対応する機械語命令に置き換える置換サブステップとを含み、

前記最適化ステップでは、前記置換サブステップで置き換えられた機械語命令を含む前記中間コードを対象として最適化を行う

ことを特徴とするプログラム。

【請求項2】 前記演算処理定義情報は、前記ソースプログラムにインクルードされるヘッダファイルであり、

前記ヘッダファイルには、データとメソッドとからなるクラスによって前記演算処理が定義され、

前記中間コード変換ステップは、前記中間コードのうち、前記ヘッダファイルで定義されたクラスを参照しているものがあるか否かを検出することによって、前記演算処理を参照するしているものがあるか否かを検出する

ことを特徴とする請求項1記載のプログラム。

【請求項3】 前記クラスは、固定小数点型を定義するクラスであり、前記検出サブステップでは、前記固定小数点型のデータを使用している中間コードを検出する

ことを特徴とする請求項2記載のプログラム。

【請求項4】 前記クラスのメソッドは、前記固定小数点型のデータを対象とする演算子を定義するメソッドであり、

前記検出サブステップでは、演算子と演算の対象となるデータの型との組が前記メソッドでの定義と一致するか否かによって前記検出をし、

前記置換サブステップでは、前記演算子と前記データの型との組が一致する中間コードを、対応する機械語命令に置き換える

ことを特徴とする請求項3記載のプログラム。

【請求項5】 前記クラスは、SIMD型を定義するクラスであり、前記検出サブステップでは、前記SIMD型のデータを使用している中間コードを検出する

ことを特徴とする請求項2記載のプログラム。

【請求項6】 前記クラスのメソッドは、前記SIMD型のデータを対象とする演算子を定義するメソッドであり、

前記検出サブステップでは、演算子と演算の対象となるデータの型との組が前記メソッドでの定義と一致するか否かによって前記検出をし、

前記置換サブステップでは、前記演算子と前記データの型との組が一致する中間コードを、対応する機械語命令に置き換える

ことを特徴とする請求項5記載のプログラム。

【請求項7】 前記クラスには、対応する処理を実現する1つの機械語命令が対応づけられ、

前記置換サブステップでは、前記中間コードを、前記クラスに対応づけられた1つの機械語命令に置き換える

ことを特徴とする請求項2～6のいずれか1項に記載のプログラム。

【請求項8】 前記クラスには、対応する処理を実現する2以上の機械語命

令が対応づけられ、

前記置換サブステップでは、前記中間コードを、前記クラスに対応づけられた2以上の機械語命令に置き換える

ことを特徴とする請求項2～6のいずれか1項に記載のプログラム。

【請求項9】 前記演算処理定義情報は、前記ソースプログラムにインクルードされるヘッダファイルであり、

前記ヘッダファイルには、関数によって前記演算処理が定義され、

前記中間コード変換ステップは、前記中間コードのうち、前記ヘッダファイルで定義された関数を参照しているものがあるか否かを検出することによって、前記演算処理を参照するしているものがあるか否かを検出する

ことを特徴とする請求項1記載のプログラム。

【請求項10】 前記関数は、対応する処理を実現する1つの機械語命令が記述された関数であり、

前記置換サブステップでは、前記中間コードを前記関数に記述された1つの機械語命令に置き換える

ことを特徴とする請求項9記載のプログラム。

【請求項11】 前記関数には、入力データの最上位ビットから0が連続するビット数を返す関数が含まれ、

前記関数に記述されている機械語命令は、第1レジスタに格納された値について、最上位ビットから0が連続するビット数を数え、その結果を第2レジスタに格納する機械語命令である

ことを特徴とする請求項10記載のプログラム。

【請求項12】 前記関数には、入力データの最上位ビットから1が連続するビット数を返す関数が含まれ、

前記関数に記述されている機械語命令は、第1レジスタに格納された値について、最上位ビットから1が連続するビット数を数え、その結果を第2レジスタに格納する機械語命令である

ことを特徴とする請求項10記載のプログラム。

【請求項13】 前記関数には、入力データの最上位ビットと同じ値が連続

するビット数を返す関数が含まれ、

前記関数に記述されている機械語命令は、第1レジスタに格納された値について、最上位ビットと同じ値が連続するビット数を数え、その結果を第2レジスタに格納する機械語命令である

ことを特徴とする請求項10記載のプログラム。

【請求項14】 前記関数は、入力データの最上位ビットと同じ値が最上位ビットの次から何ビット連続するかを返す関数であり、

前記関数に記述されている機械語命令は、第1レジスタに格納された値について、最上位ビットと同じ値が最上位ビットの次から何ビット連続するかを数え、その結果を第2レジスタに格納する機械語命令である

ことを特徴とする請求項13記載のプログラム。

【請求項15】 前記関数には、入力データ中に含まれる1のビット数を返す関数が含まれ、

前記関数に記述されている機械語命令は、第1レジスタに格納された値について、1のビット数を数え、その結果を第2レジスタに格納する機械語命令である  
ことを特徴とする請求項10記載のプログラム。

【請求項16】 前記関数には、入力データから、指定されたビット位置のビットを抽出し、符号拡張した値を返す関数が含まれ、

前記関数に記述されている機械語命令は、第1レジスタに格納された値から、第2レジスタで指定されたビット位置のビットを取り出し、符号拡張して第3レジスタに格納する機械語命令である

ことを特徴とする請求項10記載のプログラム。

【請求項17】 前記関数には、入力データから、指定されたビット位置のビットを抽出し、ゼロ拡張した値を返す関数が含まれ、

前記関数に記述されている機械語命令は、第1レジスタに格納された値から、第2レジスタで指定されたビット位置のビットを取り出し、ゼロ拡張して第3レジスタに格納する機械語命令である

ことを特徴とする請求項10記載のプログラム。

【請求項18】 前記関数は、対応する処理を実現する2以上の機械語命令

からなる機械語命令列が記述された関数であり、

前記置換サブステップでは、前記中間コードを前記関数に記述された機械語命令列に置き換える

ことを特徴とする請求項9記載のプログラム。

【請求項19】 前記関数には、モジュロアドレッシングのアドレス更新をする関数が含まれる

ことを特徴とする請求項18記載のプログラム。

【請求項20】 前記関数に記述されている機械語命令列には、第1レジスタに格納された値の所定のビットフィールドを第2レジスタに格納された値で置き換えることによって得られる値を第3レジスタに格納する機械語命令が含まれる

ことを特徴とする請求項19記載のプログラム。

【請求項21】 前記関数には、ビットリバースアドレッシングのアドレス更新をする関数が含まれる

ことを特徴とする請求項18記載のプログラム。

【請求項22】 前記関数に記述されている機械語命令列には、第1レジスタに格納された値の所定のビットフィールドをビット単位で位置反転することによって得られる値を第3レジスタに格納する機械語命令が含まれる

ことを特徴とする請求項21記載のプログラム。

【請求項23】 前記関数には、最適化におけるレジスタ割り付けの対象となる汎用レジスタに加えて、レジスタ割り付けの対象とならないアキュムレータも更新する演算であって、アキュムレータを参照型として一時変数を指定することが可能な関数が含まれる

ことを特徴とする請求項9記載のプログラム。

【請求項24】 前記関数は、乗算を行う関数であって、乗算結果を格納するアキュムレータを参照型として一時変数を指定することが可能な関数である

ことを特徴とする請求項23記載のプログラム。

【請求項25】 前記関数は、乗算を行う関数であって、乗算結果を格納するアキュムレータを参照型として一時変数を指定することが可能な関数である

ことを特徴とする請求項23記載のプログラム。

【請求項26】 前記置換サブステップでは、前記関数を参照している中間コードを、当該関数の引数の種類に対応した種類のオペランドを持つ機械語命令に置き換える

ことを特徴とする請求項9記載のプログラム。

【請求項27】 前記置換サブステップでは、

前記関数関数の引数がすべて定数である場合には、それらの定数を畳み込んで得られる定数値をオペランドを持つ機械語命令に置き換え、

前記関数関数の引数の一部が定数である場合には、即値オペランドを持つ機械語命令に置き換え、

前記関数関数の引数がすべて変数である場合には、レジスタオペランドを持つ機械語命令に置き換える

ことを特徴とする請求項26記載のプログラム。

【請求項28】 前記最適化ステップは、異なる型間の演算を行う複数の中間コード又は複数の機械語命令を、当該演算を行う1つの機械語命令に置き換える型変換最適化サブステップを含む

ことを特徴とする請求項1記載のプログラム。

【請求項29】 前記型変換サブステップでは、 $n$ ビットの2つの変数を乗算し、結果を $2n$ ビットの変数に格納する演算を行う複数の中間コード又は複数の機械語命令を、当該演算を行う1つの機械語命令に置き換える

ことを特徴とする請求項28記載のプログラム。

【請求項30】 前記型変換サブステップでは、前記2つの変数に対して、 $n$ ビットから $2n$ ビットに型変換する明示的な宣言がなされている場合に、前記演算を前記機械語命令に置き換える

ことを特徴とする請求項29記載のプログラム。

【請求項31】 前記コンパイラは、2以上の固定小数点型を対象として演算する2以上の固定小数点モードを備えるプロセッサを対象し、

前記パーサステップでは、前記固定小数点モードを切り替える旨の記述を前記ソースプログラムにおいて検出し、

前記コンパイラは、さらに、

前記パーサーステップにおいて前記固定小数点モードを切り替える旨の記述が検出された場合に、その記述に従って、固定小数点モードを切り替えるための機械語命令を挿入する固定小数点モード切替ステップを含むことを特徴とする請求項1記載のプログラム。

【請求項32】 前記固定小数点モードを切り替える旨の記述は、対象となる関数に対応づけられ、

前記固定小数点モード切替ステップでは、対応する関数の先頭及び末尾に、それぞれ、固定小数点モードの退避及び復帰に相当する機械語命令を挿入することを特徴とする請求項31記載のプログラム。

【請求項33】 前記最適化ステップは、前記ソースプログラムにおいて、特定箇所での実行時間が所定サイクル数だけ確保されるようなレイテンシを指定する旨の記述を検出し、検出した指定に従って、前記レイテンシが確保されるように機械語命令のスケジューリングを行うレイテンシ最適化サブステップを含むことを特徴とする請求項1記載のプログラム。

【請求項34】 前記レイテンシ最適化サブステップでは、第1ラベルが付された第1機械語命令と第2ラベルが付された第2機械語命令間を対象とした一定サイクル数のレイテンシを指定する旨の記述を検出すると、前記第1機械語命令が実行されてから前記第2機械語命令が実行されるまでに前記サイクル数だけ実行時間がかかるように、前記スケジューリングを行うことを特徴とする請求項33記載のプログラム。

【請求項35】 前記レイテンシ最適化サブステップでは、所定のレジスタへのアクセスを対象とした一定サイクル数のレイテンシを指定する旨の記述を検出すると、前記レジスタにアクセスする機械語命令が実行されてから次に当該レジスタにアクセスする機械語命令が実行されるまでに前記サイクル数だけ実行時間がかかるように、前記スケジューリングを行うことを特徴とする請求項33記載のプログラム。

【請求項36】 前記コンパイラは、さらに、前記演算処理定義情報で使用されている機械語命令を当該コンパイラが対象としている第1プロセッサとは異

なる第2プロセッサの機械語命令に置き換えるためのクラスライブラリを備えることを特徴とする請求項1記載のプログラム。

【請求項37】 コンパイラの対象となるソースプログラムにインクルードされるヘッダファイルが記録されたコンピュータ読み取り可能な記録媒体であつて、

請求項2～36のいずれか1項に記載されたヘッダファイルが記録された記録媒体。

【請求項38】 コンパイラの対象となるソースプログラムにインクルードされるクラスライブラリが記録されたコンピュータ読み取り可能な記録媒体であつて、

請求項36に記載されたクラスライブラリが記録された記録媒体。

【請求項39】 コンパイラの対象となるソースプログラムが記録されたコンピュータ読み取り可能な記録媒体であつて、

請求項2～36のいずれか1項に記載されたヘッダファイル及びクラスライブラリの少なくとも1つをインクルードしているソースプログラムが記録された記録媒体。

【請求項40】 ソースプログラムを機械語プログラムに翻訳するコンパイラ装置であつて、

対象とするプロセッサに特有の機械語命令に対応する演算処理が予め定義された演算処理定義情報を保持する手段と、

前記ソースプログラムを解析するパーサー手段と、

解析されたソースプログラムを中間コードに変換する中間コード変換手段と、

変換された中間コードを最適化する最適化手段と、

最適化された中間コードを機械語命令に変換するコード生成手段とを備え、

前記中間コード変換手段は、

前記中間コードのうち、前記演算処理定義情報を定義された演算処理を参照しているものがあるか否かを検出する検出部と、

あると検出された場合に、その中間コードを、対応する機械語命令に置き換える置換部とを含み、

前記最適化手段は、前記置換部で置き換えられた機械語命令を含む前記中間コードを対象として最適化を行うことを特徴とするコンパイラ装置。

【請求項41】 ソースプログラムを機械語プログラムに翻訳するコンパイル方法であって、

前記プログラムは、

前記ソースプログラムを解析するパーサーステップと、

解析されたソースプログラムを中間コードに変換する中間コード変換ステップと、

変換された中間コードを最適化する最適化ステップと、

最適化された中間コードを機械語命令に変換するコード生成ステップとを含み

前記中間コード変換ステップは、

前記中間コードのうち、対象とするプロセッサに特有の機械語命令に対応する演算処理が予め定義された演算処理定義情報において定義された演算処理を参照しているものがあるか否かを検出する検出サブステップと、

あると検出された場合に、その中間コードを、対応する機械語命令に置き換える置換サブステップとを含み、

前記最適化ステップでは、前記置換サブステップで置き換えられた機械語命令を含む前記中間コードを対象として最適化を行う

ことを特徴とするコンパイル方法。

#### 【発明の詳細な説明】

##### 【0001】

##### 【発明の属する技術分野】

本発明は、C++言語等の高級言語で記述されたソースプログラムを機械語プログラムに変換するコンパイラに関し、特に、コンパイラによる最適化に関する

##### 【0002】

##### 【従来の技術】

近年のプロセッサの高機能化に伴い、プロセッサが備える高い機能を有効に引き出すことが可能な性能の高いコンパイラが求められている。つまり、ターゲットプロセッサが備える高機能な専用命令を効率的に生成するコンパイラが求められている。

## 【0003】

例えば、デジタル信号処理等のメディア処理に必要とされる各種固定小数点フォーマットでの演算命令を実行するプロセッサや、 SIMD (Single Instruction Multiple Data) 型の命令を実行する高機能なプロセッサが開発されているが、そのようなプロセッサをターゲットとするコンパイラは、各種固定小数点フォーマットでの演算命令や SIMD 型命令を効率的に生成することによってコードサイズ及び実行速度の点で最適化を図ることが求められる。

## 【0004】

## 【発明が解決しようとする課題】

しかしながら、従来のコンパイラは、 C++ 言語等の高級言語で記述されたソースプログラムに対しては、必ずしも、プロセッサが備える高機能な専用命令を効率的に生成しているとは言えない。そのために、コードサイズと実行速度の点で厳しい条件が求められるメディア処理等のアプリケーション開発においては、ユーザは、クリティカルな箇所については、アセンブラー命令で記述せざるを得ないのが現状である。ところが、アセンブラー命令でのプログラミングは、 C++ 言語等の高級言語を用いた開発に比べ、開発工数が大きくなるだけでなく、保守性や移植性についても著しく劣るという問題がある。

## 【0005】

また、従来のコンパイラでは、プロセッサが備える高機能な専用命令を生成する等の最適化処理は、コンパイラ自身に組み込まれている。つまり、ターゲットプロセッサの特徴を有効に活用して最適化を図る処理モジュールは、コンパイラ自身に組み込まれ、一体化されている。そのために、コンパイラの機能拡張を図る場合や、ターゲットプロセッサの仕様が変更された場合等においては、コンパイラ全体の再構築が必要とされ、その度に、コンパイラのバージョンアップ等を繰り返さなければならないという問題がある。

【0006】

そこで、本発明は、このような状況に鑑みてなされたものであり、プロセッサが備える高機能な専用命令を効率的に生成することができるコンパイラを提供することを目的とする。

また、コンパイラ自身のバージョンアップを頻繁に繰り返すことなく、機能拡張等の改良を行っていくことが可能なコンパイラ等を提供することも目的とする。

【0007】

【課題を解決するための手段】

上記目的を達成するために、本発明に係るコンパイラ用プログラムは、ソースプログラムを機械語プログラムに翻訳するコンパイラ用のプログラムであって、対象とするプロセッサに特有の機械語命令に対応する演算処理が予め定義された演算処理定義情報を含み、前記プログラムは、前記ソースプログラムを解析するパーサステップと、解析されたソースプログラムを中間コードに変換する中間コード変換ステップと、変換された中間コードを最適化する最適化ステップと、最適化された中間コードを機械語命令に変換するコード生成ステップとを含み、前記中間コード変換ステップは、前記中間コードのうち、前記演算処理定義情報で定義された演算処理を参照しているものがあるか否かを検出する検出サブステップと、あると検出された場合に、その中間コードを、対応する機械語命令に置き換える置換サブステップとを含み、前記最適化ステップでは、前記置換サブステップで置き換えられた機械語命令を含む前記中間コードを対象として最適化を行うことを特徴とする。

【0008】

例えば、本発明は、ソースプログラムにインクルードされるヘッダファイルと、そのソースプログラムを機械語プログラムに翻訳するコンパイラとから構成されるプログラムであって、前記ヘッダファイルには、データとメソッドとからなるクラスが定義され、前記コンパイラは、前記ソースプログラムを解析するパーサステップと、解析されたソースプログラムを中間コードに変換する中間コード変換ステップと、変換された中間コードを最適化する最適化ステップと、最適

化された中間コードを機械語命令に変換するコード生成ステップとを含み、前記中間コード変換ステップは、前記中間コードのうち、前記ヘッダファイルで定義されたクラスを参照しているものがあるか否かを検出する検出サブステップと、あると検出された場合に、その中間コードを、対応する機械語命令に置き換える置換サブステップとを含み、前記最適化ステップでは、前記置換サブステップで置き換えられた機械語命令を含む前記中間コードを対象として最適化を行うプログラムである。

## 【0009】

ここで、前記クラスは、固定小数点型を定義するクラスであり、前記検出サブステップでは、前記固定小数点型のデータを使用している中間コードを検出してもよい。そして、前記クラスのメソッドは、前記固定小数点型のデータを対象とする演算子を定義するメソッドであり、前記検出サブステップでは、演算子と演算の対象となるデータの型との組が前記メソッドでの定義と一致するか否かによって前記検出をし、前記置換サブステップでは、前記演算子と前記データの型との組が一致する中間コードを、対応する機械語命令に置き換えてよい。

## 【0010】

また、本発明に係るコンパイラ用プログラムは、ソースプログラムにインクルードされるヘッダファイルと、そのソースプログラムを機械語プログラムに翻訳するコンパイラとから構成されるプログラムであって、前記ヘッダファイルには、関数が定義され、前記コンパイラは、前記ソースプログラムを解析するパーサーステップと、解析されたソースプログラムを中間コードに変換する中間コード変換ステップと、変換された中間コードを最適化する最適化ステップと、最適化された中間コードを機械語命令に変換するコード生成ステップとを含み、前記中間コード変換ステップは、前記中間コードのうち、前記ヘッダファイルで定義された関数を参照しているものがあるか否かを検出する検出サブステップと、あると検出された場合に、その中間コードを、対応する機械語命令に置き換える置換サブステップとを含み、前記最適化ステップでは、前記置換サブステップで置き換えられた機械語命令を含む前記中間コードを対象として最適化を行うことを特徴とする。

## 【0011】

なお、本発明は、このようなコンパイラ用プログラムとして実現することができるだけでなく、このようなプログラムに含まれるステップを手段とするコンパイラ装置として実現したり、このような特徴的なコンパイラあるいはヘッダファイルが記録されたコンピュータ読み取り可能な記録媒体として実現することもできる。そして、どのようなプログラムやデータファイルは、CD-ROM等の記録媒体やインターネット等の伝送媒体を介して広く流通させることができるのは言うまでもない。

## 【0012】

## 【発明の実施の形態】

以下、本発明に係るコンパイラの実施の形態について図面を用いて詳細に説明する。

本実施の形態におけるコンパイラは、C/C++言語等の高級言語で記述されたソースプログラムを特定のプロセッサ（ターゲット）が実行できる機械語プログラムに翻訳するクロスコンパイラであり、生成する機械語プログラムのコードサイズや実行時間に関する最適化の指示をきめ細かく指定することができるという特徴を有する。

## 【0013】

## [プロセッサ]

まず、本実施の形態におけるコンパイラの対象となるプロセッサの一例について、図1～図36を用いて説明する。

本実施の形態におけるコンパイラの対象となるプロセッサは、例えば、通常のマイコンに比べて実行可能な命令の並列性が高く、AVメディア系信号処理技術分野をターゲットとして開発された汎用プロセッサである。

## 【0014】

図1は、そのようなプロセッサの概略ブロック図の一例である。このプロセッサ1は、32ビットを基本語長（ワード）とする演算装置であり、命令制御部10、デコード部20、レジスタファイル30、演算部40、I/F部50、命令メモリ部60、データメモリ部70、拡張レジスタ部80及びI/Oインターフ

エース部90から構成される。演算部40は、SIMD型命令の演算を実行する算術論理・比較演算器41～43、乗算・積和演算器44、バレルシフタ45、除算器46及び変換器47からなる。乗算・積和演算器44は、ビット精度を落とさないように、最長で65ビットで累算する。また、乗算・積和演算器44は、算術論理・比較演算器41～43と同様、SIMD型命令の実行が可能である。更に、このプロセッサ1は、算術論理・比較演算命令が最大3並列実行可能である。

#### 【0015】

図2は、算術論理・比較演算器41～43の概略図を示す。算術論理・比較演算器41～43それぞれは、ALU部41a、飽和処理部41b及びフラグ部41cから構成される。ALU部41aは、算術演算器、論理演算器、比較器、TST器からなる。対応する演算データのビット幅は、8ビット(バイト。このときは、演算器を4並列で使用)、16ビット(ハーフワード。このときは、演算器を2並列で使用)、32ビットである(ワード。このときは、全演算器で32ビットデータ処理)。更に算術演算結果に対しては、フラグ部41c等により、オーバーフローの検出とコンディションフラグの生成が行われる。各演算器、比較器、TST器の結果は、算術右シフト、飽和処理部41bによる飽和、最大・最小値検出、絶対値生成処理が行われる。

#### 【0016】

図3は、バレルシフタ45の構成を示すブロック図である。バレルシフタ45は、セレクタ45a、45b、上位バレルシフタ45c、下位バレルシフタ45d及び飽和処理部45eから構成され、データの算術シフト(2の補数体系のシフト)または、論理シフト(符号なしシフト)を実行する。通常は、32ビットもしくは、64ビットのデータを入出力としている。レジスタ30a、30bに格納された被シフトデータに対して、別のレジスタまたは即値でシフト量が指定される。データは、左63ビット～右63ビットの算術または論理シフトが行われ、入力ビット長で出力される。

#### 【0017】

また、バレルシフタ45は、SIMD型命令に対して、8、16、32、64

ビットのデータをシフトすることができる。例えば、8ビットデータのシフトを4並列で処理することができる。

#### 【0018】

算術シフトは、2の補数体系のシフトであり、加算や減算時的小数点の位置合わせや、2のべき乗の乗算(2、2の2乗、2の(-1)乗、2の(-2)乗倍など)等のために行われる。

#### 【0019】

図4は、変換器47の構成を示すブロック図である。変換器47は、飽和ブロック(SAT)47a、BSEQブロック47b、MSKGENブロック47c、VSUMBブロック47、BCNTブロック47e及びILブロック47fから構成される。

#### 【0020】

飽和ブロック(SAT)47aは、入力データに対する飽和処理を行う。32ビットデータを飽和処理するブロックを2つ持つことにより、2並列のSIMD型命令をサポートする。

#### 【0021】

BSEQブロック47bは、MSBから連続する0か1をカウントする。

MSKGENブロック47cは、指定されたビット区間を1、それ以外を0として出力する。

VSUMBブロック47dは、入力データを指定されたビット幅に区切り、その総和を出力する。

BCNTブロック47eは、入力データで1となっているビットの数をカウントする。

ILブロック47fは、入力データを指定されたビット幅に区切り、各データブロックを入れ換えた値を出力する。

#### 【0022】

図5は、除算器46の構成を示すブロック図である。除算器46は、被除数を64ビット、除数を32ビットとし、商と剰余を32ビットずつ出力する。商と剰余を求めるまでに34サイクルを必要とする。符号付き、符号なし、両方のデータを扱うことが可能である。ただし、被除数と除数において符号の有無の設定

は共通とする。その他、オーバーフローフラグ、0除算フラグを出力する機能を有する。

#### 【0023】

図6は、乗算・積和演算器44の構成を示すブロック図である。乗算・積和演算器44は、2つの32ビット乗算器(MUL)44a、44b、3つの64ビット加算器(Adder)44c～44e、セレクタ44f及び飽和処理部(Saturation)44gから構成され、以下の乗算、積和演算を行う。

- ・ $32 \times 32$ ビットのsignedの乗算、積和、積差演算
- ・ $32 \times 32$ ビットのunsignedの乗算
- ・ $16 \times 16$ ビットの2並列のsignedの乗算、積和、積差演算
- ・ $32 \times 16$ ビットの2並列のsignedの乗算、積和、積差演算

これらの演算を整数、固定小数点フォーマット(h1、h2、w1、w2)のデータに対して行う。また、これらの演算に対し、丸め、飽和を行う。

#### 【0024】

図7は、命令制御部10の構成を示すブロック図である。命令制御部10は、命令キッシュ10a、アドレス管理部10b、命令バッファ10c～10e、ジャンプバッファ10f及びローテーション部(rotation)10gから構成され、通常時及び分岐時の命令供給を行う。128ビットの命令バッファを3つ(命令バッファ10c～10e)持つことにより、最大並列実行数に対応している。分岐処理に関しては、分岐実行前に、ジャンプバッファ10f等を介して、後述するTARレジスタに予め分岐先アドレスを格納する(settar命令)。TARレジスタに格納された分岐先アドレスを使用して、分岐を行う。

#### 【0025】

なお、本実施の形態におけるコンパイラの対象となるプロセッサ1は、例えば、VLIWアーキテクチャを持つプロセッサである。ここで、VLIWアーキテクチャとは、1つの命令語中に複数の命令(ロード、ストア、演算、分岐など)を格納し、それらを全て同時に実行するアーキテクチャである。プログラマは、並列実行可能な命令を1つの発行グループとして記述することによって、その発行グループを並列処理させることができる。本明細書では、発行グループの区切り

を";;"で示す。以下に表記例を示す。

(例1)

```
mov r1, 0x23;;
```

この命令記述は、命令`mov`のみを実行することを意味する。

(例2)

```
mov r1, 0x38
```

```
add r0, r1, r2
```

```
sub r3, r1, r2;;
```

これらの命令記述は、命令`mov`、`add`、`sub`を3並列で実行することを意味する

命令制御部10は、発行グループを識別し、デコード部20に送る。デコード部20では、発行グループの命令を解析し、必要な資源を制御する。

### 【0026】

次に、このようなプロセッサ1が備えるレジスタについて説明する。

プロセッサ1のレジスタセットは、以下の表1に示される通りである。

【表1】

レジスタ名	ビット幅	本数	用途
R0～R31	32ビット	32本	汎用レジスタ。データメモリのポインタ、演算命令におけるデータの格納等に使用します。
TAR	32ビット	1本	分岐用レジスタ。分岐時の分岐アドレスの格納に使用します。
LR	32ビット	1本	リンク用レジスタ。
SVR	16ビット	2本	退避用レジスタ。コンディションフラグ(CFR)と各種モードを退避します。
M0～M1 (MH0:ML0～MH1:ML1)	64ビット	2本	演算用レジスタ。演算命令におけるデータの格納に使用します。

### 【0027】

また、このようなプロセッサ1のフラグセット（後述する条件フラグレジスタ等で管理されるフラグ）は、以下の表2に示される通りである。

## 【表2】

C0～C7	1ビット	8本	条件フラグ。条件成立・不成立を表します。
VC0～VC3	1ビット	4本	メディア処理拡張命令用条件フラグ。条件成立・不成立を表します。
OVS	1ビット	1本	オーバーフロー・フラグ。演算時のオーバーフローを検出します。
CAS	1ビット	1本	キャリーフラグ。演算時のキャリーを検出します。
BPO	5ビット	1本	ビット位置指定。マスク処理命令時に処理対象となるビット位置を指定します。
ALN	2ビット	1本	バイトアライン指定。
FXP	1ビット	1本	固定小数点演算モード。
UDR	32ビット	1本	未定義レジスタ

## 【0028】

図8は、汎用レジスタ（R0～R31）30aの構造を示す図である。汎用レジスタ（R0～R31）30aは、実行対象となっているタスクのコンテキストの一部を構成し、データまたはアドレスを格納する32ビットのレジスタ群である。なお、汎用レジスタR30およびR31は、それぞれグローバルポインタ、スタックポインタとして、ハードウェアが使用する。

## 【0029】

図9は、リンクレジスタ（LR）30cの構造を示す図である。なお、このリンクレジスタ（LR）30cと関連して、このプロセッサ1は、図示されていない退避レジスタ（SVR）も備える。リンクレジスタ（LR）30cは、関数コール時のリターンアドレスを格納する32ビットのレジスタである。なお、退避レジスタ（SVR）は、関数コール時の条件フラグレジスタのコンディションフラグ（CFR.CF）を退避する16ビットのレジスタである。リンクレジスタ（LR）30cは、後述する分岐レジスタ（TAR）と同様に、ループ高速化にも使用される。下位1ビットは常に0が読み出されるが、書き込み時には0を書き込む必要がある。

## 【0030】

例えば、call(br1,jmp1)命令を実行した場合には、このプロセッサ1は、リンクレジスタ（LR）30cに戻りアドレスを退避し、退避レジスタ（SVR）にコンディションフラグ(CFR.CF)を退避する。また、jmp命令を実行した場合には、リンクレジスタ（LR）30cから戻りアドレス(分岐先アドレス)を取り出し

、プログラムカウンタ（PC）を復帰させる。さらに、ret(jmpr)命令を実行した場合には、リンクレジスタ（LR）30cから分岐先アドレス（戻りアドレス）を取り出し、プログラムカウンタ（PC）に格納（復帰）する。さらに、退避レジスタ（SVR）からコンディションフラグを取り出し、条件フラグレジスタ（CFR）32のコンディションフラグ領域CFR.CFに格納（復帰）する。

#### 【0031】

図10は、分岐レジスタ（TAR）30dの構造を示す図である。分岐レジスタ（TAR）30dは、分岐ターゲットアドレスを格納する32ビットのレジスタである。主に、ループの高速化に用いられる。下位1ビットは常に0が読み出されるが、書き込み時には0を書き込む必要がある。

#### 【0032】

例えば、jmp,jloop命令を実行した場合には、プロセッサ1は、分岐レジスタ（TAR）30dから分岐先アドレスを取り出し、プログラムカウンタ（PC）に格納する。分岐レジスタ（TAR）30dに格納されたアドレスの命令が分岐用命令バッファに格納されている場合は、分岐ペナルティが0になる。分岐レジスタ（TAR）30dにループの先頭アドレスを格納しておくことでループを高速化することができる。

#### 【0033】

図11は、プログラム状態レジスタ（PSR）31の構造を示す図である。プログラム状態レジスタ（PSR）31は、実行対象となっているタスクのコンテキストの一部を構成し、以下に示されるプロセッサ状態情報を格納する32ビットのレジスタである。

#### 【0034】

ビットSWE:VMP (Virtual Multi-Processor) のLP (Logical Processor) 切替えイネーブルを示す。「0」はLP切替え不許可を示し、「1」はLP切替え許可を示す。

#### 【0035】

ビットFXP:固定小数点モードを示す。「0」はモード0 (MSBとMSBから1ビット目との間に小数点があるものとして演算をするモード。以下、「\_

1系」ともいう。)を示し、「1」はモード1(MSBから1ビット目ととMSBから2ビット目との間に小数点があるものとして演算をするモード。2系ともいう。)を示す。

#### 【0036】

ビットIH：割込み処理フラグであり、マスカブル割込み処理中であることを示す。「1」は割込み処理中であることを示し、「0」は割込み処理中でないことを示す。割込みが発生すると自動的にセットされる。rti命令で割込みから復帰したところが、他の割込み処理中かプログラム処理中であるのかを見分けるために使用される。

#### 【0037】

ビットEH：エラーまたはNMIを処理中であることを示すフラグである。「0」はエラー/NMI割込み処理中でないことを示し、「1」はエラー/NMI割込み処理中であることを示す。EH=1のとき、非同期エラーまたはNMIが発生した場合は、マスクされる。また、VMPイネーブル時はVMPのプレート切り替えがマスクされる。

#### 【0038】

ビットPL[1:0]：特権レベルを示す。「00」は特権レベル0、つまり、プロセッサアブストラクションレベルを示し、「01」は特権レベル1（設定できない）を示し、「10」は特権レベル2、つまり、システムプログラムレベルを示し、「11」は特権レベル3、つまり、ユーザプログラムレベルを示す。

#### 【0039】

ビットLPIE3：LP固有割込み3イネーブルを示す。「1」は割込み許可を示し、「0」は割込み不許可を示す。

ビットLPIE2：LP固有割込み2イネーブルを示す。「1」は割込み許可を示し、「0」は割込み不許可を示す。

ビットLPIE1：LP固有割込み1イネーブルを示す。「1」は割込み許可を示し、「0」は割込み不許可を示す。

ビットLPIE0：LP固有割込み0イネーブルを示す。「1」は割込み許可を示し、「0」は割込み不許可を示す。

ビットAEE：ミスアライメント例外イネーブルを示す。「1」はミスアライメント例外許可を示し、「0」はミスアライメント例外不許可を示す。

ビットIE：レベル割込みイネーブルを示す。「1」はレベル割込み許可を示し、「0」はレベル割込み不許可を示す。

ビットIM[7:0]：割込みマスクを示す。レベル0～7まで定義され、個々のレベルでマスクすることができる。レベル0が最も高いレベルとなる。IMによりマスクされていない割込み要求のうち最も高いレベルを持った割込み要求のみがプロセッサ1に受理される。割込み要求を受理すると受理したレベル以下のレベルはハードウェアで自動的にマスクされる。IM[0]はレベル0のマスクであり、IM[1]はレベル1のマスクであり、IM[2]はレベル2のマスクであり、IM[3]はレベル3のマスクであり、IM[4]はレベル4のマスクであり、IM[5]はレベル5のマスクであり、IM[6]はレベル6のマスクであり、IM[7]はレベル7のマスクである。

reserved：予約ビットを示す。常に0が読み出される。書き込む時は0を書き込む必要がある。

#### 【0040】

図12は、条件フラグレジスタ(CFR)32の構造を示す図である。条件フラグレジスタ(CFR)32は、実行対象となっているタスクのコンテキストの一部を構成する32ビットのレジスタであり、コンディションフラグ(条件フラグ)、オペレーションフラグ(演算フラグ)、ベクタコンディションフラグ(ベクタ条件フラグ)、演算命令用ビット位置指定フィールド、SIMDデータアライン情報フィールドから構成される。

#### 【0041】

ビットALN[1:0]：アラインモードを示す。valnvc命令のアラインモードを設定する。

ビットBPO[4:0]：ビットポジションを示す。ビット位置指定の必要な命令で使用する。

ビットVC0～VC3：ベクタ条件フラグである。LSB側のバイトあるいはハーフワードから順にVC0に対応し、MSB側がVC3に対応する。

ビットOVS：オーバーフローフラグ(サマリー)である。飽和発生やオーバー

フロー検出でセットされる。検出されなかった場合は、命令実行前の値を保持する。クリアはソフトで行う必要がある。

**ビットC A S :** キャリーフラグ(サマリー)である。addc命令でキャリーまたはsubc命令でボローが発生した場合セットされる。addc命令でキャリーもしくはsubc命令でボローが発生しなかった場合は、命令実行前の値を保持する。クリアはソフトで行う必要がある。

**ビットC 0～C 7 :** コンディションフラグである。条件付き実行命令における条件 (TRUE/FALSE) を示す。条件付き命令の条件とビットC 0～C 7との対応は、命令に含まれるプレディケート・ビットによって決定される。なお、フラグC 7は常に値が1である。フラグC 7へのFALSE条件の反映(0書き込み)は無視される。

**r e s e r v e d :** 予約ビットを示す。常に0が読み出される。書き込む時は0を書き込む必要がある。

#### 【0042】

図13は、アキュムレータ(M0, M1)30bの構造を示す図である。このアキュムレータ(M0, M1)30bは、実行対象となっているタスクのコンテキストの一部を構成し、図13(a)に示される32ビットレジスタMH0-MH1(乗除算・積和用レジスタ(上位32ビット))と、図13(b)に示される32ビットレジスタML0-ML1乗除算・積和用レジスタ(下位32ビット)とからなる。

#### 【0043】

レジスタMH0-MHは、乗算命令では結果の上位32ビットを格納するのに使用される。積和命令ではアキュムレータの上位32ビットとして使用される。また、ビットストリームを取り扱う場合に汎用レジスタと組み合わせて使用することができる。レジスタML0-ML1は、乗算命令では結果の下位32ビットを格納するのに使用される。積和命令ではアキュムレータの下位32ビットとして使用される。

#### 【0044】

図14は、プログラムカウンタ(PC)33の構造を示す図である。このプログラムカウンタ(PC)33は、実行対象となっているタスクのコンテキストの

一部を構成し、実行中の命令のアドレスを保持する32ビットのカウンタである。下位1ビットは常に0が格納される。

## 【0045】

図15は、PC退避用レジスタ（IPC）34の構造を示す図である。このPC退避用レジスタ（IPC）34は、実行対象となっているタスクのコンテキストの一部を構成する32ビットのレジスタであり、下位1ビットは常に0が読み出されるが、書き込み時には0を書き込む必要がある。

## 【0046】

図16は、PSR退避用レジスタ（IPSR）35の構造を示す図である。このPSR退避用レジスタ（IPSR）35は、実行対象となっているタスクのコンテキストの一部を構成し、プログラム状態レジスタ（PSR）31を退避するための32ビットのレジスタであり、プログラム状態レジスタ（PSR）31の予約ビットに対応する部分は常に0が読み出されるが、書き込み時には0を書き込む必要がある。

## 【0047】

次に、本実施の形態におけるコンパイラの対象となるプロセッサ1のメモリ空間について説明する。例えば、プロセッサ1では、4GBのリニアなメモリ空間を32分割し、128MB単位の空間に命令SRAM（Static RAM）とデータSRAMが割り当てられる。この128MBの空間を1ブロックとして、SAR（SRAM Area Register）にアクセスしたいブロックを設定する。アクセスされたアドレスがSARで設定された空間である場合は、直接命令SRAM/データSRAMに対してアクセスを行うが、SARで設定された空間でない場合は、バスコントローラ（BCU）に対してアクセス要求を出す。BCUにはオン・チップ・メモリ（OCM）、外部メモリ、外部デバイス、I/Oポート等が接続されており、それらのデバイスに対して読み書きを行うことができる。

## 【0048】

図17は、本実施の形態におけるコンパイラの対象となるプロセッサ1のパイプライン動作を示すタイミング図である。このプロセッサ1は、本図に示されるように、例えば、基本的に命令フェッチ、命令割り当て（ディスパッチ）、デコー

ド、実行、書き込みの5段パイプラインで構成されている。

#### 【0049】

図18は、このようなプロセッサ1による命令実行時の各パイプライン動作を示すタイミング図である。命令フェッチステージでは、プログラムカウンタ（PC）33で指定されるアドレスの命令メモリをアクセスし、命令を命令バッファ10c～10e等に転送する。命令割り当てステージでは、分岐系命令に対する分岐先アドレス情報の出力、入力レジスタ制御信号の出力、可変長命令の割り当てを行い、命令をインストラクションレジスタ（IR）に転送する。デコードステージでは、IRをデコード部20に入力し、演算器制御信号、メモリアクセス信号を出力する。実行ステージでは、演算を実行、演算結果をデータメモリか汎用レジスタ（R0～R31）30aに出力する。書き込みステージでは、データ転送、演算結果を汎用レジスタに格納する。

#### 【0050】

本実施の形態におけるコンパイラの対象となるプロセッサ1は、例えば、VL IWアーキテクチャにより上記の処理を最高3並列で行うことができる。したがって、図18に示された動作については、本プロセッサ1は、図19に示されるタイミングで並列に実行する。

#### 【0051】

次に、以上のように構成されたプロセッサ1の命令セットの例について説明する。

以下の表3～表5は、本実施の形態におけるコンパイラの対象となるプロセッサ1が実行する命令をカテゴリー別に分類した表である。

【表3】

カテゴリー	演算器	命令オペコード
メモリ転送命令(ロード)	M	ld, ldh, ldhu, ldb, ldbu, ldp, ldhp, ldbp, ldbh, ld buh, ldbhp, ldbuhp
メモリ転送命令(ストア)	M	st, sth, stb, stp, sthp, stbp, stbh, stbhp
メモリ転送命令(その他)	M	dpref, ldstb
外部レジスタ転送命令	M	rd, rde, wt, wte
分岐命令	B	br, brl, call, jmp, jmp1, jmpf, ret, jmpf, jloop, s etbb, setlr, settar
ソフトウェア割込み命令	B	rti, pi0, pi01, pi1, pi11, pi2, pi21, pi3, pi31, pi 4, pi41, pi5, pi51, pi6, pi61, pi7, pi71, sc0, sc1, sc2, sc3, sc4, sc5, sc6, sc7
VMP/割込み制御命令	B	intd, inte, vmpsleep, vmpsus, vmpswd, vmpswe, vmp wait
算術演算命令	A	abs, absvh, absvw, add, addarvw, addc, addmsk, ad ds, addsr, addu, addvh, addvw, neg, negvh, negvw, rsub, s1add, s2add, sub, subc, submsk, subs, subv h, subvw, max, min
論理演算命令	A	and, andn, or, sethi, xor, not
比較命令	A	cmpCC, cmpCCa, cmpCCn, cmpCCo, tstn, tstna, tstd n, tstno, tstz, tstza, tstdn, tstdzo
転送命令	A	mov, movcf, mvclcas, mvclows, setlo, vcchk
NOP命令	A	nop
シフト命令1	S1	asl, aslvh, aslvw, asr, asrvh, asrvw, lsl, ls r, ro l, ror
シフト命令2	S2	aslp, aslpvw, asrp, asrpvw, lslp, ls rp

【表4】

カテゴリー	演算器	命令オペコード
抽出命令	S2	ext, extb, extbu, exth, exthu, extr, extru, extu
マスク命令	C	msk, mskgen
飽和命令	C	sat12, sat9, satb, satbu, sath, satw
変換命令	C	valn, valn1, valn2, valn3, valnvc1, valnvc2, valnvc3, valnvc4, vhpkb, vhpkh, vhunpkb, vhunpkh, vintlh, vintlhh, vintlib, vintlh, vlpkb, vlpku, vlpkh, vlpkhu, vlpkhu, vlpkbu, vlpkbu, vlpkh, vlpkhu, vstovb, vstovh, vunpk1, vunpk2, vxchngh, vexth
ビットカウント命令	C	bcnt1, bseq, bseq0, bseq1
その他	C	byterev, extw, mskbrvb, mskbrvh, rndvh, movp
乗算命令1	X1	fmulhh, fmulhhr, fmulhw, fmulhww, hmul, lmul
乗算命令2	X2	fmulww, mul, mulu
積和命令1	X1	fmachh, fmachhr, fmachw, fmachww, hmac, lmac
積和命令2	X2	fmacww, mac
積差命令1	X1	fmsuhh, fmsuhhr, fmsuhw, fmsuww, hmsu, lmsu
積差命令2	X2	fmsuww, msu
割算命令	DIV	div, divu
デバッグ命令	DBGM	dbgm0, dbgm1, dbgm2, dbgm3

【表5】

カテゴリー	演算器	命令オペコード
SIMD 算術演算命令	A	vabshvh, vaddb, vaddh, vaddhvc, vaddvhv, vaddrh vc, vaddsb, vaddsh, vaddsrh, vaddsrh, vasubb, vc chk, vhaddh, vhaddhv, vhsuh, vhsuhvh, vladhh .vladdhv, vlsuh, vlsuhvh, vnegb, vnegh, vneg vhv, vsaddb, vsaddh, vsgnh, vssuh, vssuh, vs subb, vssuh, vsubb, vsubh, vsubhv, vsubsh, vsu mh, vsumh2, vsumrh2, vxaddh, vxaddhv, vxsubh, v xsubhv, vmaxb, vmaxh, vminb, vminh, vmoht, vsel
SIMD 比較命令	A	vcmpqeb, vcmpqeh, vcmppgb, vcmppgh, vcmpptb, vc mpgth, vcmpleb, vcmpleh, vcmpltb, vcmplth, vcm pneb, vcmppneh, vscmpqeb, vscmpqeh, vscmpgpb, vscmpgeh, vscmpg tb, vscmpgth, vscmpleb, vscmpleh, vscmpltb, vsc mplth, vscmpneb, vscmpneh
SIMD シフト命令1	S1	vaslb, vaslh, vaslvh, vasrb, vasrh, vasrvh, vls b, vlslh, vlsrb, vlsrh, vrolb, vrolh, vrorb, vror h
SIMD シフト命令2	S2	vasl, vaslvw, vasr, vasrvw, vls1, vlsr
SIMD 鮑和命令	C	vsath, vsath12, vsath8, vsath8u, vsath9
SIMD その他の命令	C	vabssumb, vrndvh
SIMD 乗算命令	X2	vfmulh, vfmulhr, vfmulw, vhfmulh, vhfmulhr, vhf mulw, vhmul, vlfmulh, vlfmulhr, vlfmulw, vlmul, vmul, vpfmulhww, vxfmulh, vxfmulhr, vxfmulw, vx mul
SIMD 積和命令	X2	vfmach, vfmachr, vfmacw, vhfmach, vhfmachr, vhf macw, vhmac, vlfmach, vlfmachr, vlfmacw, vlmac, vmac, vpfmachww, vxfmach, vxfmachr, vxfmacw, vx mac
SIMD 積差命令	X2	vfmusu, vfmusu, vhfmsuh, vhfmsuw, vhmsu, vlfmsu h, vlfmsuw, vimsu, vmsu, vxfmsuh, vxfmsuw, vxmsu

## 【0052】

なお、表中の「演算器」は、その命令が使用する演算器を示す。演算器の略号の意味は次の通りである。つまり、「A」はALU命令、「B」は分岐命令、「C」は変換命令、「DIV」は除算命令、「DBG」はデバッグ命令、「M」はメモリアクセス命令、「S1」、「S2」はシフト命令、「X1」、「X2」は乗算命令を意味する。

## 【0053】

図20は、このようなプロセッサ1が実行する命令のフォーマットの例を示す図である。そのフォーマットには、図20(a)に示される16ビット命令コ

ーマットと、図20（b）に示される32ビット命令フォーマットとがある。

#### 【0054】

なお、図中における略号の意味は次の通りである。つまり、「E」はエンドビット（並列実行の境界）、「F」はフォーマットビット（00、01、10：16ビット命令フォーマット、11：32ビット命令フォーマット）、「P」はプレディケート（実行条件：8個の条件フラグC0～C7のいずれかを指定）、「OP」はオペコードフィールド、「R」はレジスタフィールド、「I」は即値フィールド、「D」ディスプースメントフィールドを意味する。なお、「E」フィールドはVLIWに特有のもので、E=0の命令は次の命令と並列に実行される。つまり、「E」フィールドによって並列度が可変のVLIWを実現している。

#### 【0055】

図21～図36は、プロセッサ1が実行する命令の概略的な機能を説明する図である。つまり、図21は、カテゴリー「ALUadd（加算）系」に属する命令を説明する図であり、図22は、カテゴリー「ALUsub（減算）系」に属する命令を説明する図であり、図23は、カテゴリー「ALUlogic（論理演算）系ほか」に属する命令を説明する図であり、図24は、カテゴリー「CMP（比較演算）系」に属する命令を説明する図であり、図25は、カテゴリー「mul（乗算）系」に属する命令を説明する図であり、図26は、カテゴリー「mac（積和演算）系」に属する命令を説明する図であり、図27は、カテゴリー「msu（積差演算）系」に属する命令を説明する図であり、図28は、カテゴリー「MEM1d（メモリ読み出し）系」に属する命令を説明する図であり、図29は、カテゴリー「MEMstore（メモリ書き出し）系」に属する命令を説明する図であり、図30は、カテゴリー「BRA（分岐）系」に属する命令を説明する図であり、図31は、カテゴリー「BSas1（算術バallelシフト）系ほか」に属する命令を説明する図であり、図32は、カテゴリー「BSlsr（論理バallelシフト）系ほか」に属する命令を説明する図であり、図33は、カテゴリー「CNVvaln（算術変換）系」に属する命令を説明する図であり、図34は、カテゴリー「CNV（一般変換）系」に属する命令を説明する図であり、図35は、カテゴリー「SATvlpk（飽和処理）系」に属する命令を説明する図であり、図36は、カテゴリー「ETC（その他）系」に属する命

令を説明する図である。

## 【0056】

これらの図において、項目「SIMD」は、その命令の型（SISD (SINGLE) か SIMD かの区別）を示し、項目「サイズ」は、演算の対象となる個々のオペランドのサイズを示し、項目「命令」は、その命令のオペコードを示し、項目「オペランド」は、その命令のオペランドを示し、項目「CFR」は、条件フラグレジスタの変化を示し、項目「PSR」は、プロセッサ状態レジスタの変化を示し、項目「代表的な動作」は、動作の概要を示し、項目「演算器」は、使用される演算器を示し、項目「3116」は、命令のサイズを示す。

## 【0057】

以下に、後述する具体例で使用される主な命令に対するプロセッサ1の動作を説明する。

**andn Rc,Ra,Rb**

RaとRbの反転の論理積をとり、Rcに格納する。

**asl Rb,Ra,I5**

Raを即値(I5)ビット数だけ算術左シフトする。

**and Rb,Ra,I8**

Raと値(I8)の論理積をとり、Rbに格納する。

**bseq0 Rb,Ra**

RaのMSBから連続する 0 の数を数えて、Rbに格納する。

## 【0058】

**bseq1 Rb,Ra**

RaのMSBから連続する 1 の数を数えて、Rbに格納する。

**bseq Rb,Ra**

RaのMSBの 1 bit 下から連続する符号ビットの数を数えて、Rbに格納する。Raが 0 の場合は、0 を出力する。

**bcnt1 Rb,Ra**

Raの 1 の数を数えて、Rbに格納する。

**extr Rc,Ra,Rb**

Rbでbit位置を指定し、Raの内容の一部を取り出し、符号拡張してRcに格納する。

## 【0059】

**extru Rc,Ra,Rb**

Rbでbit位置を指定し、Raの内容の一部を取り出し、符号拡張せずにRcに格納する。

**fmulhh Mm,Rc,Ra,Rb**

Ra,Rb,Rcを 16bit 値として扱い、Mm（乗算用アキュムレータ）を32bit値として扱う。RaとRbを固定小数点乗算する。結果をMmとRcに格納する。結果が符号付き 32 bitで表せない場合飽和する。

**fmulhw Mm,Rc,Ra,Rb**

RaとRbを 16bit 値として扱い、Mm,Rcを32bit値として扱う。RaとRbを固定小数点乗算する。結果をMmとRcに格納する。結果が符号付き 32 bitで表せない場合飽和する。

**mul Mm,Rc,Ra,Rb**

RaとRbを整数乗算する。結果をMmとRcに格納する。

## 【0060】

**mac Mm,Rc,Ra,Rb,Mn**

RaとRbを整数乗算し、Mnと加算する。結果をMmとRcに格納する。

**mov Rb,Ra**

RaをRbに転送する。

**or Rc,Ra,Rb**

RaとRbの論理和をとり、Rcに格納する。

**rde C0:C1,Rb,(Ra)**

Raを外部レジスタ番号として、その外部レジスタの値をRbに読み込む。読み込みの成功・失敗をC0とC1（それぞれ、条件フラグ）に出力し、失敗時は拡張レジスタエラー例外が発生する。

**wte C0:C1,(Ra),Rb**

Raを外部レジスタ番号として、Rbの値をその外部レジスタに書き出す。書き

出しの成功・失敗をC0とC1に出力し、失敗時は拡張レジスタエラー例外が発生する。

**vaddh Rc,Ra,Rb**

各レジスタをハーフワードベクタ形式で扱う。RaとRbを加算(SIMD straight)する。

#### 【0061】

##### [コンパイラ]

次に、以上のプロセッサ1をターゲットとする本実施の形態におけるコンパイラについて説明する。

図37は、本実施の形態におけるコンパイラ100の構成を示す機能ブロック図である。このコンパイラ100は、C/C++言語等の高級言語で記述されたソースプログラム101を、上述のプロセッサ1をターゲットプロセッサとする機械語プログラム105に変換するクロスコンパイラであり、パーソナルコンピュータ等のコンピュータ上で実行されるプログラムによって実現され、大きく分けて、パーサー部110と、中間コード変換部120と、最適化部130と、コード生成部140とから構成される。

#### 【0062】

なお、本コンパイラ100には、上記プロセッサ1に特有の専用命令を効率的に生成するためのヘッダファイル（演算子定義ファイル102及び組み込み関数定義ファイル103）が準備されている。ユーザは、ソースプログラム101において、これらのヘッダファイルをインクルードすることで、プロセッサ1に特化した（最適化された）機械語プログラム105を得ることができる。

#### 【0063】

演算子定義ファイル102は、図38～図68のリスト例に示されるように、固定小数点及びSIMD型のデータを対象とする演算子を定義したクラスを定義しているヘッダファイルである。図38～図40は、ヘッダファイルのうち、主に、モード0（\_1系）の16ビット固定小数点のデータを対象とする演算子（オペレータ）を定義している箇所のリストであり、図41～図42は、ヘッダファイルのうち、主に、モード0（\_1系）の32ビット固定小数点のデータを対

象とする演算子（オペレータ）を定義している箇所のリストであり、図4 3～図4 5は、ヘッダファイルのうち、主に、モード1（\_2系）の16ビット固定小数点のデータを対象とする演算子（オペレータ）を定義している箇所のリストであり、図4 5～図4 7は、ヘッダファイルのうち、主に、モード1（\_2系）の32ビット固定小数点のデータを対象とする演算子（オペレータ）を定義している箇所のリストであり、図4 8～図6 8は、その他の関数を定義している箇所のリストである。

## 【0064】

組み込み関数定義ファイル103は、図6 9～図7 2のリスト例に示されるように、プロセッサ1に特有の機械語命令に置き換えるための各種演算を行う関数を定義しているヘッダファイルである。図6 9～図7 1は、ヘッダファイルのうち、1つの機械語命令に置き換える関数を定義している箇所のリストであり、図7 2は、ヘッダファイルのうち、2以上の機械語命令（機械語命令列）に置き換える関数を定義している箇所のリストである。

## 【0065】

なお、これらの定義ファイル102及び103において、`asm(...){...}(...)`は、最適化asmと呼ばれる組み込みアセンブラー命令であり、以下の通り、処理される。つまり、最適化asm文の記述形式は、

## 【0066】

```
asm(<<ロード式並び>>){  
    <<最適化制御情報>>  
    <<命令指定部>>  
}(<<ストア式並び>>);
```

## 【0067】

である。ここで、「ロード式並び」は、ロード式を記述する箇所であり、「ロード式」は、レジスタにC言語での変数や四則演算などの式の結果を格納するための式であり、「レジスタ指定識別子 = 代入式」のように記述され、右辺に示される値を左辺に示される識別子に転送することを意味し、「ストア式並び」は、ストア式を記述する箇所であり、「ストア式」は、「単項式 = レジスタ指定

識別子」のように記述され、単項式で表される左辺値にレジスタ指定識別子で表されるレジスタの値を代入することを意味する。

#### 【0068】

パーサー部110は、コンパイルの対象となるソースプログラム101（インクルードされるヘッダファイルを含む）に対して、予約語（キーワード）等を抽出して字句解析する前置処理部であり、通常のコンパイラーが備える解析機能に加えて、固定小数点についてのモードの切り替えをサポートする固定小数点モード切替部111を有する。固定小数点モード切替部111は、ソースプログラム101中に、固定小数点モードを退避・復帰させるプラグマ指令（例えば、#pragma \_save\_fxpmode func等）を検出すると、プロセッサ1のPSR31のビットFXPの退避・復帰を行う機械語命令を生成する。これによって、ユーザは、固定小数点のモード0及びモード1での演算が混在したプログラミングが可能となる。

#### 【0069】

なお、「プラグマ（又は、プラグマ指令）」とは、ソースプログラム101中にユーザが任意に指定（配置）することができるコンパイラー100への指示であり、「#pragma」で始まる文字列である。

#### 【0070】

中間コード変換部120は、パーサー部110から渡されたソースプログラム101の各ステートメントを中間コードに変換する処理部であり、中間コード生成部121と機械語命令置換部122とから構成される。中間コード生成部121は、ソースプログラム101の各ステートメントを一定規則に基づいて中間コードに変換する。ここで、中間コードは、典型的には、関数呼び出しの形式で表現されるコード（例えば、「+(int a, int b)」を示すコード；「整数aに整数bを加算する」ことを示す。）である。ただし、中間コードには、このような関数呼び出し形式のコードだけでなく、プロセッサ1の機械語命令も含まれる。

#### 【0071】

機械語命令置換部122は、中間コード生成部121で生成された中間コードのうち、関数呼び出し形式の中間コードに対して、演算子定義ファイル102及

び組み込み関数定義ファイル103を参照することで、それらの定義ファイルで定義された演算子（演算対象のデータの型も含む）又は組み込み関数に一致した中間コードについては、内部に保持する置換テーブル122a又はこれらの定義ファイルで定義されたアセンブラー命令に従って、対応する機械語命令（又は機械語命令列）に置き換え、最適化部130に出力する。これによって、これらの中間コードについては、関数呼び出しの形式ではなく、機械語命令の形式で最適化部130に渡されるので、最適化部130での各種最適化が可能となる。

#### 【0072】

なお、置換テーブル122aは、予め予約された演算子による演算及び関数に対応する機械語命令（又は機械語命令列）を格納したテーブルである。また、機械語命令置換部122は、中間コード生成部121から渡された中間コードのうち、機械語命令については、そのまま最適化部130に出力する。

#### 【0073】

最適化部130は、中間コード変換部120から出力された中間コードのうち、機械語命令について、命令結合、冗長除去、命令並べ替え、レジスタ割り付け等の処理を行うことで、（1）実行速度の向上を優先した最適化、（2）コードサイズの削減を優先した最適化、（3）実行速度とコードサイズの両方の最適化、の中からユーザによって選択された種類の最適化を実行する処理部であり、一般的な最適化（「ループアンローリング」、「if変換」、「ペアメモリアクセス命令の生成」など）に加えて、本コンパイラ100に特有の最適化を行う処理部（引数最適化部131、型変換最適化部132及びレイテンシ最適化部133）を有する。

#### 【0074】

引数最適化部131は、組み込み関数（extr、extruなど）の引数に応じて、適切な命令又はシーケンス（アルゴリズム）を生成する処理部である。例えば、引数がすべて定数の場合には、それら定数を畳み込んで得られる定数値をオペランドとする機械語命令を生成し、引数の一部が定数の場合には、即値オペランドの機械語命令を生成し、引数のすべてが変数の場合には、レジスタオペランドの命令列を生成する。

## 【0075】

型変換最適化部132は、ソースプログラム101中の一定表記に基づいて、異なる型間の演算を効率化する処理部である。例えば、16ビットのデータと16ビットのデータとの乗算結果を32ビットのデータとして保持しておきたい場合に、ソースプログラム101において一定表記がなされているときには、そのような型変換を伴った乗算を行う1つの機械語命令（「fmulhw」など）を生成する。

## 【0076】

レイテンシ最適化部133は、ソースプログラム101に組み込まれたアセンブラー命令中のレイテンシに関する指示（サイクル数の指定）に基づいて、特定の区間あるいは特定の動作が指定されたサイクル数だけ実行時間がかかるよう機械語命令を並べる。これによって、従来であればプログラマが必要個数のnop命令を挿入していた作業が不要になるとともに、nop命令以外の他の機械語命令を挿入することによる最適化が可能となる。

## 【0077】

なお、「ループアンローリング」とは、ループのイタレーション（繰り返し）を展開し、複数のイタレーションを同時に実行させるために、ペアメモリアクセス命令(ldp/stp/ldhp/sthp等)を生成し、これによって、ループの並列実行可能性を向上させる最適化である。また、「if変換」とは、条件付き実行機構の命令（命令に含まれる条件（プレディケート）がプロセッサ1の状態（コンディションフラグ）と一致している場合にだけ実行される命令）を生成することにより分岐構造を除去する最適化である。また、「ペアメモリアクセス命令の生成」とは、ペアレジスタ(2本の連続するレジスタ)を対象にメモリアクセスする命令(ldp/stp/ldhp/sthp等)を生成する最適化である。

## 【0078】

また、最適化部130は、関数呼び出し形式の中間コードのうち、機械語命令に展開されないものについては、上述のような機械語命令レベルでの最適化処理を施すことができないので、そのままコード生成部140に出力する。

## 【0079】

コード生成部140は、最適化部130から出力された中間コード（関数呼び出し形式のコード及び最適化された機械語命令を含む）に対して、内部に保持する変換テーブル等を参照することで、全てのコードを機械語命令に置き換えることで、機械語プログラム105を生成する。

#### 【0080】

次に、以上のように構成されたコンパイラ100の特徴的な動作について、具体的な例を示しながら説明する。

図73は、機械語命令置換部122の動作を示すフローチャートである。機械語命令置換部122は、中間コード生成部121で生成された中間コードのうち、関数呼び出し形式のコードについて、演算子定義ファイル102で定義された演算子（演算対象のデータ型を含む）及び組み込み関数定義ファイル103で定義された関数と一致するか否かを判断し（ステップS101）、一致する場合には（ステップS101でYes）、内部に保持する置換テーブル122a又はこれらの定義ファイル102及び103で定義されたアセンブラー命令にしたがって、機械語命令に置き換える（ステップS102）という処理を繰り返す（ステップS100～S103）。

#### 【0081】

具体的には、演算子定義ファイル102での定義により、異なる型間の暗黙の型変換のルール等が規定され（コンストラクタの定義による）、4種類の固定小数点の型、つまり、

- ・ 「FIX16\_1」；第14ビットと第15ビット（MSB）との間に小数点を持つ符号付き16ビットと、
- ・ 「FIX16\_2」；第13ビットと第14ビットとの間に小数点を持つ符号付き16ビットと、
- ・ 「FIX32\_1」；第30ビットと第31ビット（MSB）との間に小数点を持つ符号付き32ビットと、
- ・ 「FIX32\_2」；第29ビットと第30ビットとの間に小数点を持つ符号付き32ビットと

が定義されているので、機械語命令置換部122は、例えば、

## 【0082】

```
FIX16_1 a, b, c;
```

```
c = a * b;
```

というソースプログラムを、

**fmulhh m0,Rc,Ra,Rb** (固定小数点乗算命令)

という機械語命令に置き換える。

## 【0083】

これによって、ユーザは、FIX16\_1、FIX16\_2、FIX32\_1、FIX32\_2という型について、通常のコンパイラにおける基本型と同様に宣言し、使用することができる。そして、生成された機械語命令は、周辺のコードと含めて、最適化部130における命令結合、冗長除去、命令並べ替え、レジスタ割り付け等の最適化の対象となり、最適化が施され得る。

## 【0084】

同様に、演算子定義ファイル102での定義により、異なる型間の暗黙の型変換のルール等が規定され（コンストラクタの定義による）、4種類のSIMD型つまり、

- ・「VINT8x4」；8ビット整数データの4並列と、
- ・「VINT16x2」；16ビット整数データの2並列と、
- ・「VFIX161x2」；モード0（\_\_1系）の16ビット固定小数点データの2並列と、
- ・「VFIX162x2」；モード1（\_\_2系）の16ビット固定小数点データの2並列と

が定義されているので、機械語命令置換部122は、例えば、

## 【0085】

```
VINT16x2 a, b, c;
```

```
c = a + b;
```

というソースプログラムを、

**vaddh Rc,Ra,Rb** (SIMD型加算命令)

という機械語命令に置き換える。

## 【0086】

これによって、ユーザは、VINT8x2、VINT16x2、VFIX161x2、VFIX162x2という型について、通常のコンパイラにおける基本型と同様に宣言し、使用することができる。そして、生成された機械語命令は、周辺のコードと含めて、最適化部130における命令結合、冗長除去、命令並べ替え、レジスタ割り付け等の最適化の対象となり、最適化が施され得る。

## 【0087】

また、組み込み関数定義ファイル103においては、プロセッサ1が備える高機能命令を使用することが可能な関数（例えば、「\_abs(a)」等）とそれに対応する高機能命令（例えば、1つの機械語命令「abs Rb, Ra」等）が定義されているので、機械語命令置換部122は、例えば、

## 【0088】

```
b = _abs(a);
```

というソースプログラムを、

```
abs Rb,Ra
```

という機械語命令に置き換える。

## 【0089】

これによって、ユーザは、複雑な処理を、C++言語やアセンブラー命令で作成することなく、予め用意された組み込み関数を呼び出すだけで、実現することができる。そして、生成された機械語命令は、周辺のコードと含めて、最適化部130における命令結合、冗長除去、命令並べ替え、レジスタ割り付け等の最適化の対象となり、最適化が施され得る。

## 【0090】

同様に、また、組み込み関数定義ファイル103において、プロセッサ1が備える高機能命令を使用することが可能な関数（例えば、「\_div(a, b)」等）とそれに対応する高機能命令（例えば、機械語命令列「extw, aslp, div」等）が定義されているので、機械語命令置換部122は、例えば、

## 【0091】

```
c = _div(a, b);
```

というソースプログラムを、

```
extw    Mn,Rc,Ra
aslp    Mn,Rc,Mn,Rc,15
div     MHm,Rc,MHn,Rc,Rb
```

という機械語命令列に置き換える。

#### 【0092】

これによって、ユーザは、複雑な処理を、C++言語やアセンブラー命令で作成することなく、予め用意された組み込み関数を呼び出すだけで、実現することができる。そして、生成された機械語命令列は、周辺のコードと含めて、最適化部130における命令結合、冗長除去、命令並べ替え、レジスタ割り付け等の最適化の対象となり、最適化が施され得る。

#### 【0093】

なお、組み込み関数定義ファイル103に列挙されている組み込み関数のうち、（1）1つの機械語命令に変換される関数、（2）2以上の機械語命令（機械語命令列）に変換される関数、及び、（3）レジスタ割り付けの対象とならない資源（アキュムレータなど）を指定することが可能な関数の代表的なもの（特に、メディア処理に有効なもの）は、以下の通りである。

#### 【0094】

（1）1つの機械語命令に変換される組み込み関数

- ・「\_bseq0(x)」：

入力の最上位ビットから何ビット0が連続するかを検出する関数である。書式は、以下の通りである。

```
int _bseq0(FIX16_1 val) // 0をカウント
int _bseq0(FIX16_2 val) // 0をカウント
int _bseq0(FIX32_1 val) // 0をカウント
int _bseq0(FIX32_2 val) // 0をカウント
```

これらの関数は、「val」中の連続する0の個数（ビット数）をカウントした値を返す。これらの関数に対応する機械語命令は、組み込み関数定義ファイル103に定義されている通りである。

## 【0095】

- ・「\_bseq1(x)」：

入力の最上位ビットから何ビット1が連続するかを検出する関数である。書式は、以下の通りである。

```
int _bseq1(FIX16_1 val) // 1をカウント
int _bseq1(FIX16_2 val) // 1をカウント
int _bseq1(FIX32_1 val) // 1をカウント
int _bseq1(FIX32_2 val) // 1をカウント
```

これらの関数は、「val」中の連続する1の個数（ビット数）をカウントした値が返す。これらの関数に対応する機械語命令は、組み込み関数定義ファイル103に定義されている通りである。

## 【0096】

- ・「\_bseq(x)」：

入力の最上位ビットと同じ値が最上位ビットの次から何ビット連続するかを検出する関数である。書式は、以下の通りである。

```
int _bseq(FIX16_1 val)
int _bseq(FIX16_2 val)
int _bseq(FIX32_1 val)
int _bseq(FIX32_2 val)
```

これらの関数は、「val」の正規化ビット数を返す。これらの関数に対応する機械語命令は、組み込み関数定義ファイル103に定義されている通りである。

## 【0097】

- ・「\_bcnt1(x)」：

入力の全ビット中に何ビット1が含まれているかを検出する関数である。書式は、以下の通りである。

```
int _bcnt1(FIX16_1 val)
int _bcnt1(FIX16_2 val)
int _bcnt1(FIX32_1 val)
int _bcnt1(FIX32_2 val)
```

これらの関数は、「val」中の1の個数をカウントした値を返す。これらの関数に対する機械語命令は、組み込み関数定義ファイル103に定義されている通りである。

## 【0098】

- ・「\_extr(a,i1,i2)」：

入力の所定のビット位置を抽出して符号拡張する関数である。  
関数である。書式は、以下の通りである。

```
int _extr (FIX16_1 val1, int val2, int val3)
int _extr (FIX16_2 val1, int val2, int val3)
int _extr (FIX32_1 val1, int val2, int val3)
int _extr (FIX32_2 val1, int val2, int val3)
```

これらの関数は、ビット位置val2からビット位置val3で示されたval1のビットフィールドを抽出し、符号拡張を行った結果を返す。これらの関数に対する機械語命令は、組み込み関数定義ファイル103に定義されている通りである。

## 【0099】

- ・「\_extru(a,i1,i2)」：

入力の所定のビット位置を抽出してゼロ拡張する関数である。書式は、以下の通りである。

```
unsigned int _extru (FIX16_1 val, int val2, int val3)
unsigned int _extru (FIX16_2 val, int val2, int val3)
unsigned int _extru (FIX32_1 val, int val2, int val3)
unsigned int _extru (FIX32_2 val, int val2, int val3)
```

これらの関数は、ビット位置val2からビット位置val3で示されたval1のビットフィールドを抽出し、ゼロ拡張を行った結果を返す。これらの関数に対する機械語命令は、組み込み関数定義ファイル103に定義されている通りである。

## 【0100】

(2) 2以上の機械語命令（機械語命令列）に変換される組み込み関数

- ・「\_modulo\_add()」：

モジュロアドレッシングのアドレス更新を行う関数である。書式は、以下の通

りである。

```
_modulo_add(void *addr, int imm, int mask, size_t size, void *base)
```

ここで、各引数の意味は以下の通りである。

**addr:** 更新前のアドレス又はアドレス下位(モジュロ部分)

**imm:** 加算値(データ数)

**mask:** マスク幅(モジュロ幅)

**size:** データのサイズ(2のべき乗)

**base:** ベースアドレス(配列の先頭アドレス)

この関数は、モジュロアドレッシングにより、アドレスaddrから加算値immだけ加算した結果を返す。

#### 【0101】

この関数に対応する機械語命令は、組み込み関数定義ファイル103に定義されている通りである。つまり、モジュロアドレッシング計算のため、第一の入力の所定のビットフィールドを第二の入力の所定のビットフィールドで置き換える命令(addmsk)を活用している関数である。使用例は、以下の通りである。

```
int array [MODULO];
p = array;
for (i = 0; i < 100; i++) {
    *q++ = *p;
    p = (int *)_modulo_add(p, 1, N, sizeof(int), array);
}
```

ここで、変数MODULOは2のべき乗( $2^N$ )である。この使用例では、配列arrayの100個の要素を MODULO\*SIZEバイトアライメントして配置している。

#### 【0102】

- 「\_brev\_add()」：

ビットリバースアドレッシングのアドレス更新を行う関数である。書式は、以下の通りである。

```
_brev_add(void *addr, int cnt, int imm, int mask, size_t size, void *base)
```

ここで、各引数の意味は以下の通りである。

**addr:** 更新前のアドレス

**cnt:** ビットリバースカウンタ

**mm:** 加算値（データ数）

**mask:** マスク幅（リバース幅）

**size:** データのサイズ                   ※ 2のべき乗

**base:** ベースアドレス（配列の先頭アドレス）

この関数は、ビットリバースアドレッシングにより、ビットリバースカウンタ

**cnt**に対応するアドレス**addr**から加算値*mm*だけ加算した結果を返す。

#### 【0103】

この関数に対応する機械語命令は、組み込み関数定義ファイル103に定義されている通りである。つまり、ビットリバースアドレッシング計算のため、第一の入力の所定のビットフィールドに対してビット単位の位置反転を行なう命令(**mskbrvh**)を活用している関数である。使用例は、以下の通りである。

```
int array[BREV];
p = array;
for (i = 0; i < 100; i++) {
    *q++ = *p;
    p = (int *)_rev_add(p, i, 1, N, sizeof(int), array);
}
```

ここで、変数BREVは2のべき乗 ( $2^N$ ) である。この使用例では、配列arrayの100個の要素を BREV\*SIZEバイトアライメントして配置する。

#### 【0104】

(3) レジスタ割り付けの対象とならない資源（アキュムレータなど）を指定することが可能な関数

組み込み関数定義ファイル103には、最適化のおけるレジスタ割り付けの対象となる資源である汎用レジスタに加えて、レジスタ割り付けの対象とならない資源である（暗黙の資源である）アキュムレータも更新する演算(乗算、積和)であって、アキュムレータを参照型として一時変数を指定することが可能な組み込

み関数（乗算「\_mul」、積和「\_mac」）が用意されている。具体的な書式は、それぞれ、以下の通りである。

#### 【0105】

```
_mul(long &mh, long &m1, FIX16_1 &c, FIX16_1 a, FIX16_1 b);
```

この関数は、変数 a 変数 b とを乗算し、結果の 64 bit データの上位 32 ビットを乗算用上位アキュムレータ MH に、下位 32 ビットを乗算用下位アキュムレータ ML に設定し、さらに、アキュムレータ MH の下位 16 ビットとアキュムレータ ML の上位 16 ビットとを連結した 32 ビットデータを変数 c にも設定する。

```
_mac(long &mh, long &m1, FIX16_1 &c, FIX16_1 a, FIX16_1 b);
```

この関数は、変数 a と変数 b とを乗算した結果に、乗算用上位アキュムレータ MH と乗算用下位アキュムレータ ML とを連結した 64 ビットデータを加算し、結果の 64 bit データの上位 32 ビットを乗算用上位アキュムレータ MH に、下位 32 ビットを乗算用下位アキュムレータ ML に設定し、さらに、乗算用上位アキュムレータ の下位 16 ビットと乗算用下位アキュムレータ ML の上位 16 ビットとを連結した 32 ビットデータを変数 c にも設定する。

#### 【0106】

使用例は、以下の通りである。

```
long mh,m1;  
  
_mul(mh,m1,dummy,a,b);  
  
_mac(mh,m1,e,c,d);
```

というソースプログラムに対して、機械語命令置換部 122 は、組み込み関数定義ファイル 103 での定義に従って、

```
mul m0,Rx,Ra,Rb  
mov r0,mh0  
mov r1,mh1  
mov mh0,r0  
mov mh1,r1  
mac m0,Re,Rc,Rd,m0
```

という機械語命令列に置き換える。なお、上記機械語命令列のうち、第1～第3行が関数\_mulに対応するものであり、第4～第6行が関数\_macに対応するものである。このような機械語命令列は、最適化部130において、冗長除去により、第2～第5行が削除され、

```
mul m0,Rx,Ra,Rb
```

```
mac m0,Re,Rc,Rd,m0
```

という機械語命令列に最適化される。このように、レジスタ割り付けの対象とならない資源（アキュムレータなど）を指定することが可能な組み込み関数が使用された場合には、その資源に対する定義（値の格納）と使用（値の参照）との組がコンパイラ内部の最適化（最適化部130）にて削除される可能性が高いので、このような組み込み関数は、最適化の点においても有効である。

#### 【0107】

次に、本コンパイラ100の特徴的な動作のうち、最適化部130による動作を説明する。

図74は、最適化部130の引数最適化部131の動作を示すフローチャートである。引数最適化部131は、組み込み関数(extr, extruなど)の引数に応じて、適切な命令又はシーケンス(アルゴリズム)を生成するために、その関数の引数がすべて定数の場合には(ステップS110で左)、それら定数を畳み込んで得られる定数值をオペランドとする機械語命令を生成し(ステップS111)、引数の一部が定数の場合には(ステップS110で中央)、即値オペランドの機械語命令を生成し(ステップS112)、引数のすべてが変数の場合には(ステップS110で右)、レジスタオペランドの命令列を生成する(ステップS113)。

#### 【0108】

例えば、

```
d = _extru(0xffff, 7, 4);
```

のように、引数のすべて定数の場合には、

```
movRd,0xf
```

のように、畳み込んで定数值を設定した機械語命令を生成する。

一方、

```
d = _extru(a, 7, 4);
```

のように、一部の引数が定数の場合には、

```
extruRd,Ra,7,4
```

のように、即値オペランドの命令を生成する。

さらに一方、

```
d = _extru(a, b, c);
```

のように、すべての引数が変数の場合には、

```
aslRe,Rb,8
```

```
andRf,Rc,0x1f
```

```
orRg,Re,Rf
```

```
extruRd,Ra,Rg
```

のように、レジスタオペランドの命令列を生成する。

#### 【0109】

このように、引数最適化部131により、1つの組み込み関数は、常に同じ機械語命令が固定的に生成されるのではなく、その引数の性質に応じて最適化された機械語命令（又は機械語命令列）が生成されることになる。

#### 【0110】

図75は、最適化部130の型変換最適化部132の動作を説明するための演算木を示す図である。型変換最適化部132は、異なる型間の演算を効率化するために、ソースプログラム中の一定表記の演算に対して、型変換を伴う機械語命令（fmulhwなど）を生成する。

#### 【0111】

通常のC言語では、16bit × 16bitの結果の型は16bitである。16bit × 16bit -> 32bit の命令は存在するが、次のように、2つの機械語命令が生成されてしまう。例えば、

```
f32 = f16 * f16;
```

の記述に対して、

```
fmulhh // 16bit × 16bit -> 16bit
```

`as1 // 16bit -> 32bitの型変換`

の2命令が生成されてしまう。

#### 【0112】

そこで、型変換最適化部132は、ソースプログラムにおいて、`(FIX32)16bit * (FIX32)16bit`と記述されている場合には、通常であれば、図75(a)に示される演算木が生成されてしまう（型変換のコードが生成されてしまう）ところ、この演算木を図75(b)に示される演算木に変換することにより、`16bit * 16bit --> 32bit`の1命令(`fmulhw`)を生成する。

#### 【0113】

図76は、レイテンシ最適化部133の動作を説明するためのサンプルプログラムの例を示す図である。レイテンシ最適化部133は、ソースプログラム101に組み込まれたアセンブラー命令（最適化asm文）でのレイテンシに関する指示（サイクル数の指定）に基づいて、特定の区間あるいは特定の動作において、指定されたサイクル数だけ実行時間がかかるように、機械語命令をスケジューリングする。

#### 【0114】

ユーザは、2種類の指定方法によって、レイテンシを設定することができる。

その1つの方法は、図76(a)に示されるプログラムにおける指定(`LATENCY L1, L2, 2;`)のように、特定の命令に付けられたラベル間のレイテンシを指定する方法である。図76(a)の例では、レイテンシ最適化部133は、プロセッサ1によって命令wteが実行されてから命令rdeが実行されるまでに2サイクルだけ経過するように、機械語命令列の配置をスケジューリングする。

#### 【0115】

他の1つの方法は、図76(b)に示されるプログラムにおける指定（命令wte内における`LATENCY(2)`）のように、拡張レジスタ部80にアクセスする命令(`r d, wt, rde, wte`)について、次に拡張レジスタ部80にアクセスするまでのレイテンシを指定する方法である。図76(b)の例では、レイテンシ最適化部133は、プロセッサ1によって命令wteが実行され、拡張レジスタ部80にアクセスされてから再び拡張レジスタ部80にアクセスされるまでに2サイクルだけ

経過するように、機械語命令列の配置をスケジューリングする。

#### 【0116】

このようなレイテンシの設定よって、`inline`展開先のコードとの間で最適化(命令結合、冗長除去、命令並べ替え、レジスタ割り付け)が可能になるとともに、指定した命令間又はアクセス間でのレイテンシが確保される。つまり、従来であれば、ユーザは、明示的に`nop`命令を挿入しなければならないところ、このコンパイラ100によれば、必要な命令又はアクセスについて必要なレイテンシを指定するだけで済む。

#### 【0117】

図77は、パーサー部110の固定小数点モード切替部111の動作を説明するための図である。

固定小数点モード切替部111は、ソースプログラム101中に、固定小数点モードを退避・復帰させるプラグマ指令（例えば、`#pragma _save_fxpmode func`等）を検出すると、プロセッサ1のPSR31のビットFXPの退避・復帰を行う機械語命令を生成する。

#### 【0118】

なお、前提となる固定小数点に関する仕様として、\_1系(FIX16\_1, FIX32\_1)と\_2系(FIX16\_2, FIX32\_2)の型が存在し、ハードウェア（プロセッサ1）では、PSR31の1ビット(FXP)でモード切り替えをしており、さらに、関数内には単一の系統しか使用できないという条件が存在する。

#### 【0119】

そこで、これら2つの体系をプログラム上で切り替えて使用する方法として、他の体系からcallされる可能性のある関数にプラグマ(`#pragma _save_fxpmode 関数名`)を指定することとしている。これによって、固定小数点モード切替部111は、当該関数の先頭と末尾にFIX型モードの退避・復帰に相当するコードを挿入する。また、各関数のFIX型宣言をサーチし、その関数をいずれのFIX型モードでコンパイルするかを決定し、モード設定のコードを挿入する。

#### 【0120】

図77(a)は、そのプラグマ指令を伴う関数の例を示している。図77(a)

) の右側に記されたコメントが固定小数点モード切替部111による挿入処理であり、その具体的な処理は、図77(b)に示される通りである。

#### 【0121】

このようなプログラマ指令の適用例は、図77(c)に示される通りである。例えば、4つの関数、f11, f21, f22, f23について、関数f11: \_1系 は 関数f21: \_2系をcallし、関数f21: \_2系 は 関数f22: \_2系をcallし、関数f22: \_2系 は 関数f23: \_2系をcallする場合、他のモードからcallされる可能性がある関数はf21のみなので、この関数についてのみプログラマ指定するだけで、正常なモード切り替えが可能となる。

#### 【0122】

以上のように、本実施の形態におけるコンパイラ100によれば、演算子定義ファイル102及び組み込み関数定義ファイル103と機械語命令置換部122との連携処理により、ユーザは、モード0及び1の固定小数点型を通常の型として宣言し、使用することができるとともに、高級言語のレベルで組み込み関数を呼び出すことで、プロセッサ1が備える高機能な機械語命令を効率的に生成させることができる。

#### 【0123】

また、引数最適化部131による組み込み関数の引数についての最適化により、効率的なオペランドを持つ機械語命令が生成される。また、型変換最適化部132による型変換についての最適化により、型変換を伴う演算は、プロセッサ1が備える高機能な1つの機械語命令に変換される。さらに、レイテンシ最適化部133による機械語命令のスケジューリングにより、ユーザは、nop命令を挿入することなく、特定の命令間あるいは拡張レジスタへのアクセスにおけるレイテンシを確保することができる。

#### 【0124】

以上、本発明に係るコンパイラについて、実施の形態に基づいて説明したが、本発明は、この実施の形態に限定されるものではない。

例えば、本実施の形態では、固定小数点の型は、16ビット又は32ビットで、かつ、小数点がMSB又はその下位桁に位置したが、このようなフォーマットだ

けに限られず、8ビットや64ビットの固定小数点や、他の桁に小数点が位置する型を対象としてもよい。

#### 【0125】

また、ユーザに対する開発支援ツールとして、クラスライブラリを活用した動作検証手法を提供してもよい。つまり、通常であれば、図78(a)に示されるように、本実施の形態におけるターゲットマシン（プロセッサ1）用のクロスコンパイラ（コンパイラ100）を用いてテストソースと定義ファイル102及び103等をコンパイルし、得られたプロセッサ1用の機械語プログラムを専用のシミュレータで実行させることによって動作検証を行うが、これに代えて、図78(b)に示されるように、開発用ホストマシン（例えば、米国インテル社製のプロセッサ）を対象としたクラスライブラリ（演算子定義ファイル102及び組み込み関数定義ファイル103それぞれをプロセッサ1ではなく、ホストマシンの機械語命令に対応させる定義ファイル）を用意し、テストソース、定義ファイル102及び103と共に、ネイティブコンパイラ（Visual C++(R)等）でコンパイルし、得られた機械語プログラムをそのままホストマシンで実行させてもよい。これによって、慣れた環境で高速にシミュレーションし、動作検証を行うことが可能となる。

#### 【0126】

さらに、本実施の形態では、ターゲットプロセッサに特有の機械語命令に対応する演算子や組み込み関数は、ヘッダファイル（定義ファイル102及び103）として供給されたが、本発明に係るコンパイラは、このような定義ファイルでの情報をコンパイラ自身の中に組み込んだ構成としてもよい。つまり、本発明に係るコンパイラは、上記定義ファイル102及び103が組み込まれた一体形式のプログラムであって、対象とするプロセッサに特有の機械語命令に対応する演算処理が予め定義された演算処理定義情報を含み、ソースプログラムを解析するパーサステップと、解析されたソースプログラムを中間コードに変換する中間コード変換ステップと、変換された中間コードを最適化する最適化ステップと、最適化された中間コードを機械語命令に変換するコード生成ステップとを含み、中間コード変換ステップは、前記中間コードのうち、前記演算処理定義情報で定

義された演算処理を参照しているものがあるか否かを検出する検出サブステップと、あると検出された場合に、その中間コードを、対応する機械語命令に置き換える置換サブステップとを含み、前記最適化ステップでは、前記置換サブステップで置き換えられた機械語命令を含む前記中間コードを対象として最適化を行う構成としてもよい。これによって、ユーザは、ソースプログラムにおいて定義ファイルをインクルードする必要がなくなる。

#### 【0127】

##### 【発明の効果】

以上の説明から明らかのように、本発明に係るコンパイラ用プログラムは、ソースプログラムを機械語プログラムに翻訳するコンパイラ用のプログラムであって、対象とするプロセッサに特有の機械語命令に対応する演算処理が予め定義された演算処理定義情報を含み、前記プログラムは、前記ソースプログラムを解析するパーサーステップと、解析されたソースプログラムを中間コードに変換する中間コード変換ステップと、変換された中間コードを最適化する最適化ステップと、最適化された中間コードを機械語命令に変換するコード生成ステップとを含み、前記中間コード変換ステップは、前記中間コードのうち、前記演算処理定義情報で定義された演算処理を参照しているものがあるか否かを検出する検出サブステップと、あると検出された場合に、その中間コードを、対応する機械語命令に置き換える置換サブステップとを含み、前記最適化ステップでは、前記置換サブステップで置き換えられた機械語命令を含む前記中間コードを対象として最適化を行うことを特徴とする。

#### 【0128】

例えば、本発明に係るプログラムは、ソースプログラムにインクルードされるヘッダファイルと、そのソースプログラムを機械語プログラムに翻訳するコンパイラとから構成されるプログラムであって、前記ヘッダファイルには、データとメソッドとからなるクラスが定義され、前記コンパイラは、前記ソースプログラムを解析するパーサーステップと、解析されたソースプログラムを中間コードに変換する中間コード変換ステップと、変換された中間コードを最適化する最適化ステップと、最適化された中間コードを機械語命令に変換するコード生成ステッ

プとを含み、前記中間コード変換ステップは、前記中間コードのうち、前記ヘッダファイルで定義されたクラスを参照しているものがあるか否かを検出する検出サブステップと、あると検出された場合に、その中間コードを、対応する機械語命令に置き換える置換サブステップとを含み、前記最適化ステップでは、前記置換サブステップで置き換えられた機械語命令を含む前記中間コードを対象として最適化を行うプログラムである。

#### 【0129】

これによって、ソースプログラムにおいて、ヘッダファイルで定義されているクラスを参照しているステートメントがある場合には、そのステートメントに対応する中間コードは、機械語命令に変換された後に最適化処理の対象となるので、周辺の機械語命令と含めて最適化が施され得る。また、コンパイラは、コンパイラ自身の機能（最適化処理）だけでなく、ヘッダファイルでの定義との連携によって最適化を図っているので、ヘッダファイルの内容を更新するだけで、最適化の対象となるステートメントを増加させたり、最適化のレベルを向上させたりすることができる。

#### 【0130】

ここで、前記クラスは、固定小数点型を定義するクラスであり、前記検出サブステップでは、前記固定小数点型のデータを使用している中間コードを検出してよい。そして、前記クラスのメソッドは、前記固定小数点型のデータを対象とする演算子を定義するメソッドであり、前記検出サブステップでは、演算子と演算の対象となるデータの型との組が前記メソッドでの定義と一致するか否かによって前記検出をし、前記置換サブステップでは、前記演算子と前記データの型との組が一致する中間コードを、対応する機械語命令に置き換えてよい。

#### 【0131】

これによって、ヘッダファイルで定義されている固定小数点型及び演算子についても、通常の型と同様に、コンパイラによって対応する中間コードや機械語命令に変換されるので、ユーザは、ソースプログラムにおいてそのヘッダファイルをインクルードしておくだけで、通用の型と同様に、ターゲットプロセッサ特有の固定小数点モードに対応した型の宣言や使用をすることが可能となる。

## 【0132】

また、本発明に係るコンパイラ用プログラムは、ソースプログラムにインクルードされるヘッダファイルと、そのソースプログラムを機械語プログラムに翻訳するコンパイラとから構成されるプログラムであって、前記ヘッダファイルには、関数が定義され、前記コンパイラは、前記ソースプログラムを解析するパーサーステップと、解析されたソースプログラムを中間コードに変換する中間コード変換ステップと、変換された中間コードを最適化する最適化ステップと、最適化された中間コードを機械語命令に変換するコード生成ステップとを含み、前記中間コード変換ステップは、前記中間コードのうち、前記ヘッダファイルで定義された関数を参照しているものがあるか否かを検出する検出サブステップと、あると検出された場合に、その中間コードを、対応する機械語命令に置き換える置換サブステップとを含み、前記最適化ステップでは、前記置換サブステップで置き換えられた機械語命令を含む前記中間コードを対象として最適化を行うことを特徴とする。

## 【0133】

これによって、ソースプログラムにおいて、ヘッダファイルで定義されている関数（組み込み関数）を参照しているステートメントがある場合には、そのステートメントに対応する中間コードは、そのヘッダファイルで定義されている機械語命令に変換された後に最適化処理の対象となるので、周辺の機械語命令と含めて最適化が施され得る。また、ユーザは、プロセッサ専用の高機能命令を使用したい場合には、ソースプログラムにおいてそのヘッダファイルをインクルードしておくとともに、必要な組み込み関数を呼び出す旨の記述をしておくだけで済む。つまり、アセンブラー命令でコーディングすることから解放される。

## 【0134】

以上のように、本発明に係るコンパイラにより、ターゲットプロセッサが備える高機能な専用命令が効率的に生成され、高いレベルで最適化が施されるとともに、ヘッダファイルによる機能拡張等の柔軟な対応が可能となり、特に、コードサイズと実行速度の両面で厳しい仕様が求められるメディア処理アプリケーションの開発ツールとして、その実用的価値は極めて高い。

【図面の簡単な説明】

【図1】 本発明に係るコンパイラの対象となるプロセッサの概略ブロック図である。

【図2】 同プロセッサの算術論理・比較演算器の概略図を示す。

【図3】 同プロセッサのバーレルシタの構成を示すブロック図である。

【図4】 同プロセッサの変換器の構成を示すブロック図である。

【図5】 同プロセッサの除算器の構成を示すブロック図である。

【図6】 同プロセッサの乗算・積和演算器の構成を示すブロック図である

【図7】 同プロセッサの命令制御部の構成を示すブロック図である。

【図8】 同プロセッサの汎用レジスタ（R0～R31）の構造を示す図である。

【図9】 同プロセッサのリンクレジスタ（LR）の構造を示す図である。

【図10】 同プロセッサの分岐レジスタ（TAR）の構造を示す図である

【図11】 同プロセッサのプログラム状態レジスタ（PSR）の構造を示す図である。

【図12】 同プロセッサの条件フラグレジスタ（CFR）の構造を示す図である。

【図13】 同プロセッサのアキュムレータ（M0, M1）の構造を示す図である。

【図14】 同プロセッサのプログラムカウンタ（PC）の構造を示す図である。

【図15】 同プロセッサのPC退避用レジスタ（IPC）の構造を示す図である。

【図16】 同プロセッサのPSR退避用レジスタ（IPSR）の構造を示す図である。

【図17】 同プロセッサのパイプライン動作を示すタイミング図である。

【図18】 同プロセッサによる命令実行時の各パイプライン動作を示すタ

イミング図である。

【図19】 同プロセッサの並列動作を示す図である。

【図20】 同プロセッサが実行する命令のフォーマットを示す図である。

【図21】 カテゴリー「ALUadd（加算）系」に属する命令を説明する図である。

【図22】 カテゴリー「ALUsub（減算）系」に属する命令を説明する図である。

【図23】 カテゴリー「ALUlogic（論理演算）系ほか」に属する命令を説明する図である。

【図24】 カテゴリー「CMP（比較演算）系」に属する命令を説明する図である。

【図25】 カテゴリー「mul（乗算）系」に属する命令を説明する図である。

【図26】 カテゴリー「mac（積和演算）系」に属する命令を説明する図である。

【図27】 カテゴリー「msu（積差演算）系」に属する命令を説明する図である。

【図28】 カテゴリー「MEMld（メモリ読み出し）系」に属する命令を説明する図である。

【図29】 カテゴリー「MEMstore（メモリ書き出し）系」に属する命令を説明する図である。

【図30】 カテゴリー「BRA（分岐）系」に属する命令を説明する図である。

【図31】 カテゴリー「BSasl（算術バレルシフト）系ほか」に属する命令を説明する図である。

【図32】 カテゴリー「BSlsr（論理バレルシフト）系ほか」に属する命令を説明する図である。

【図33】 カテゴリー「CNVvaln（算術変換）系」に属する命令を説明する図である。

【図34】 カテゴリー「CNV（一般変換）系」に属する命令を説明する図である。

【図35】 カテゴリー「SATvlpk（飽和処理）系」に属する命令を説明する図である。

【図36】 カテゴリー「ETC（その他）系」に属する命令を説明する図である。

【図37】 本発明に係るコンパイラの構成を示す機能ブロック図である。

【図38】 演算子定義ファイルのリストの一部を示す図である。

【図39】 演算子定義ファイルのリストの一部を示す図である。

【図40】 演算子定義ファイルのリストの一部を示す図である。

【図41】 演算子定義ファイルのリストの一部を示す図である。

【図42】 演算子定義ファイルのリストの一部を示す図である。

【図43】 演算子定義ファイルのリストの一部を示す図である。

【図44】 演算子定義ファイルのリストの一部を示す図である。

【図45】 演算子定義ファイルのリストの一部を示す図である。

【図46】 演算子定義ファイルのリストの一部を示す図である。

【図47】 演算子定義ファイルのリストの一部を示す図である。

【図48】 演算子定義ファイルのリストの一部を示す図である。

【図49】 演算子定義ファイルのリストの一部を示す図である。

【図50】 演算子定義ファイルのリストの一部を示す図である。

【図51】 演算子定義ファイルのリストの一部を示す図である。

【図52】 演算子定義ファイルのリストの一部を示す図である。

【図53】 演算子定義ファイルのリストの一部を示す図である。

【図54】 演算子定義ファイルのリストの一部を示す図である。

【図55】 演算子定義ファイルのリストの一部を示す図である。

【図56】 演算子定義ファイルのリストの一部を示す図である。

【図57】 演算子定義ファイルのリストの一部を示す図である。

【図58】 演算子定義ファイルのリストの一部を示す図である。

【図59】 演算子定義ファイルのリストの一部を示す図である。

【図60】 演算子定義ファイルのリストの一部を示す図である。

【図61】 演算子定義ファイルのリストの一部を示す図である。

【図62】 演算子定義ファイルのリストの一部を示す図である。

【図63】 演算子定義ファイルのリストの一部を示す図である。

【図64】 演算子定義ファイルのリストの一部を示す図である。

【図65】 演算子定義ファイルのリストの一部を示す図である。

【図66】 演算子定義ファイルのリストの一部を示す図である。

【図67】 演算子定義ファイルのリストの一部を示す図である。

【図68】 演算子定義ファイルのリストの一部を示す図である。

【図69】 組み込み関数定義ファイルのリストの一部を示す図である。

【図70】 組み込み関数定義ファイルのリストの一部を示す図である。

【図71】 組み込み関数定義ファイルのリストの一部を示す図である。

【図72】 組み込み関数定義ファイルのリストの一部を示す図である。

【図73】 機械語命令置換部の動作を示すフローチャートである。

【図74】 最適化部の引数最適化部の動作を示すフローチャートである。

【図75】 最適化部の型変換最適化部の動作を説明するための演算木を示す図である。

【図76】 レイテンシ最適化部の動作を説明するためのサンプルプログラムの例を示す図である。

【図77】 パーサー部の固定小数点モード切替部の動作を説明するための図である。

【図78】 クラスライブラリを活用した動作検証手法を説明するための図である。

【符号の説明】

1	プロセッサ
10	命令制御部
20	デコード部
30	レジスタファイル
31	プログラム状態レジスタ (P S R)

- 3 2 条件フラグレジスタ (C F R)
- 3 3 プログラムカウンタ (P C)
- 3 4 P C 退避用レジスタ (I P C)
- 3 5 P S R 退避用レジスタ (I P S R)
- 4 0 演算部
  - 4 1 ~ 4 3 算術論理・比較演算器
  - 4 4 積和演算器
  - 4 5 バレルシフタ
  - 4 6 除算器
  - 4 7 変換器
- 5 0 I / F 部
- 6 0 命令メモリ部
- 7 0 データメモリ部
- 8 0 拡張レジスタ部
- 9 0 I / O インターフェース部
- 1 0 0 コンパイラ
- 1 0 1 ソースプログラム
- 1 0 2 演算子定義ファイル
- 1 0 3 組み込み関数定義ファイル
- 1 0 5 機械語プログラム
- 1 1 0 パーサー部
- 1 1 1 固定小数点モード切替部
- 1 2 0 中間コード変換部
- 1 2 1 中間コード生成部
- 1 2 2 機械語命令置換部
- 1 2 2 a 置換テーブル
- 1 3 0 最適化部
- 1 3 1 引数最適化部
- 1 3 2 型変換最適化部

特2002-226682

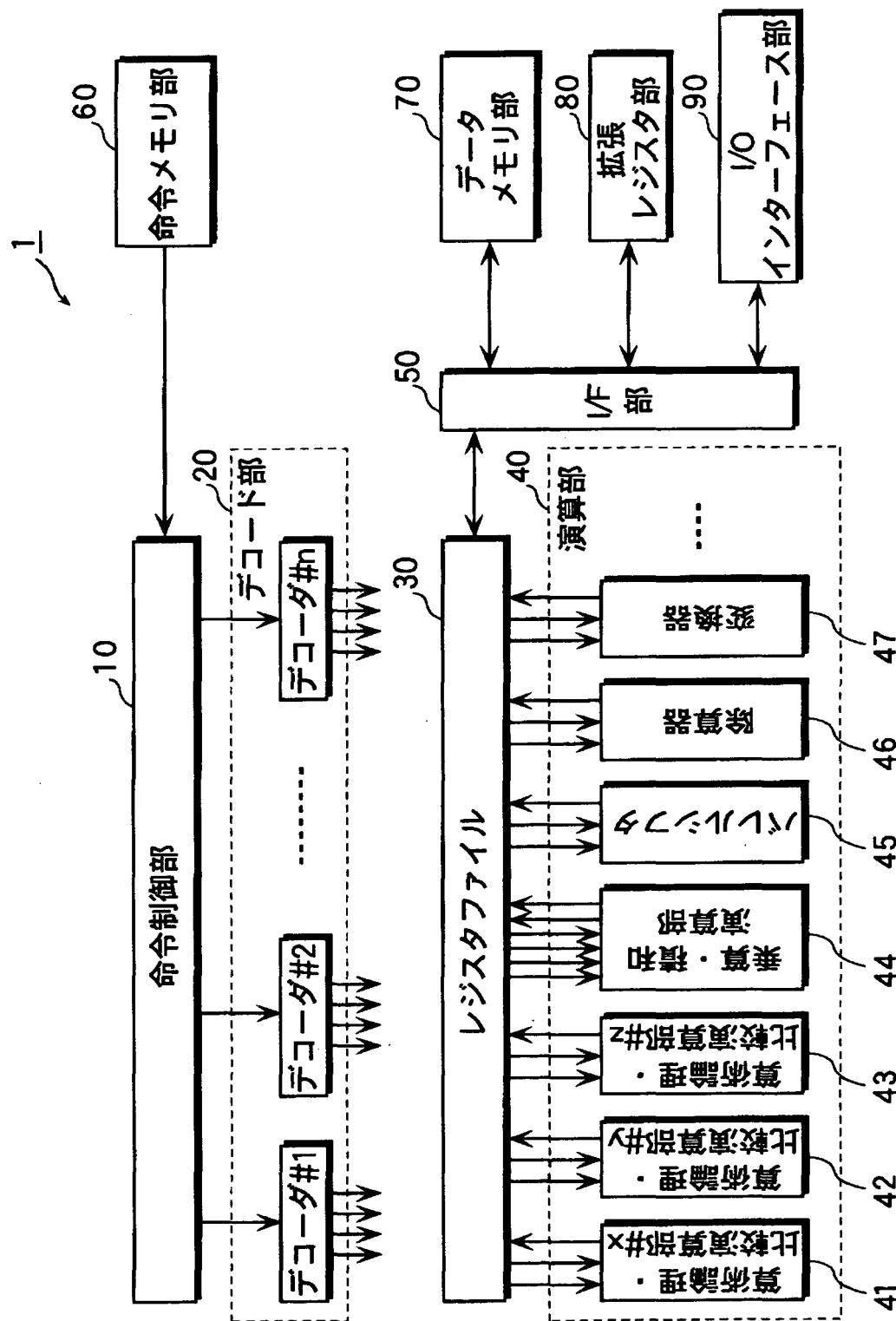
133 レイテンシ最適化部

140 コード生成部

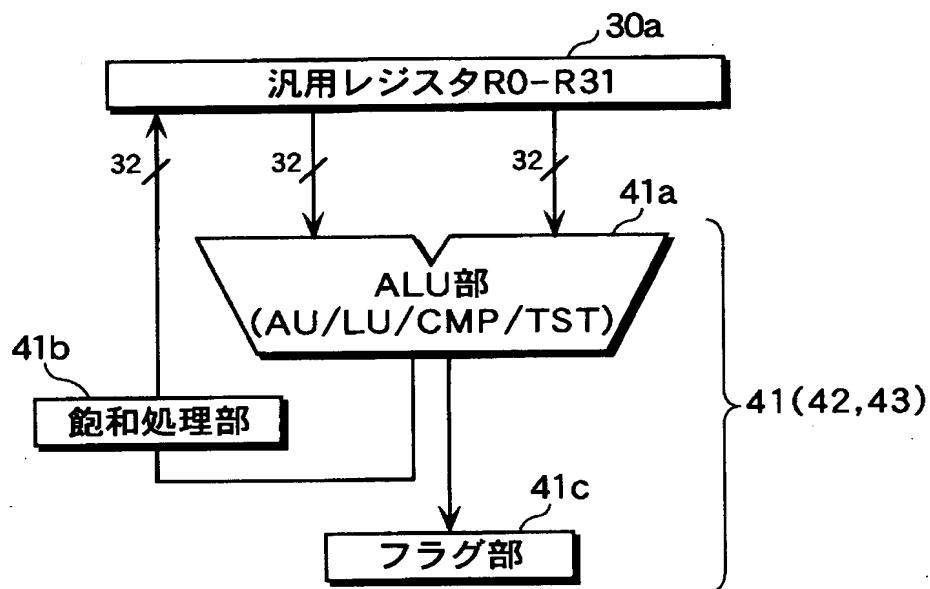
【書類名】

図面

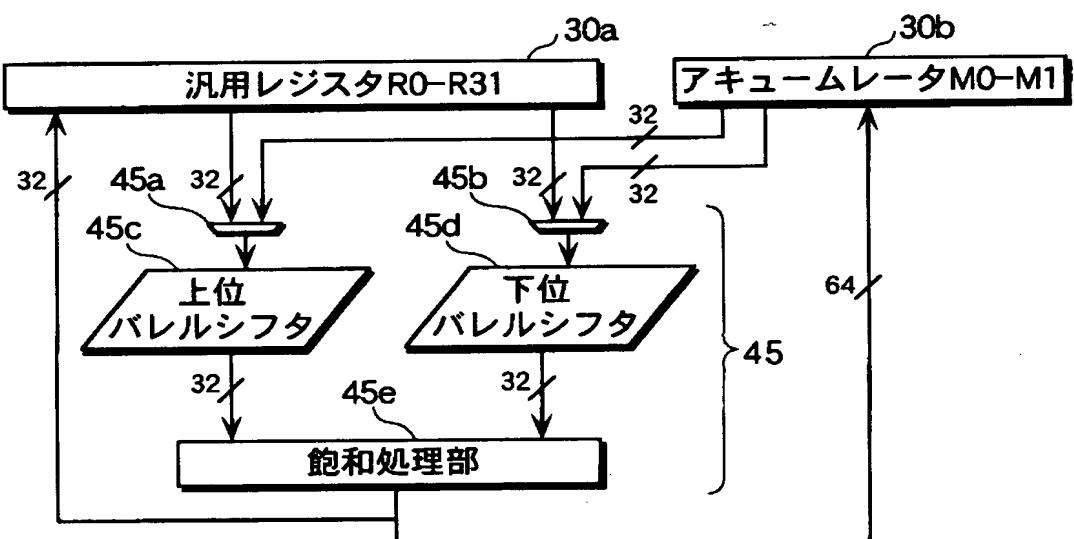
【図1】



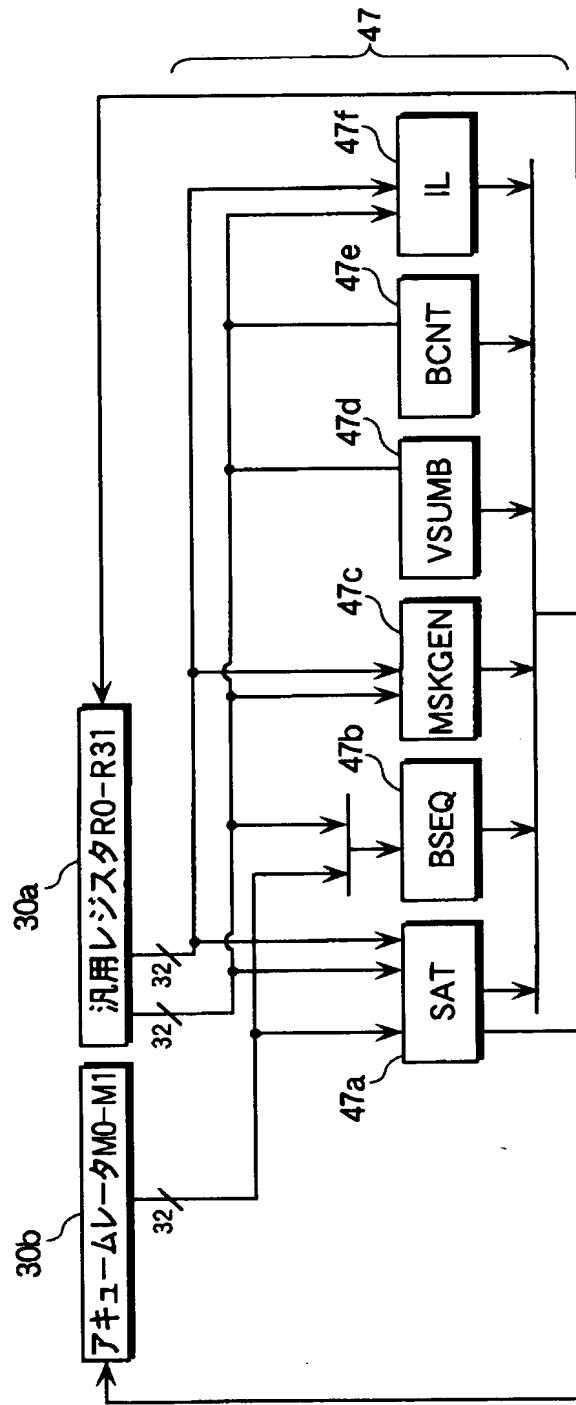
【図2】



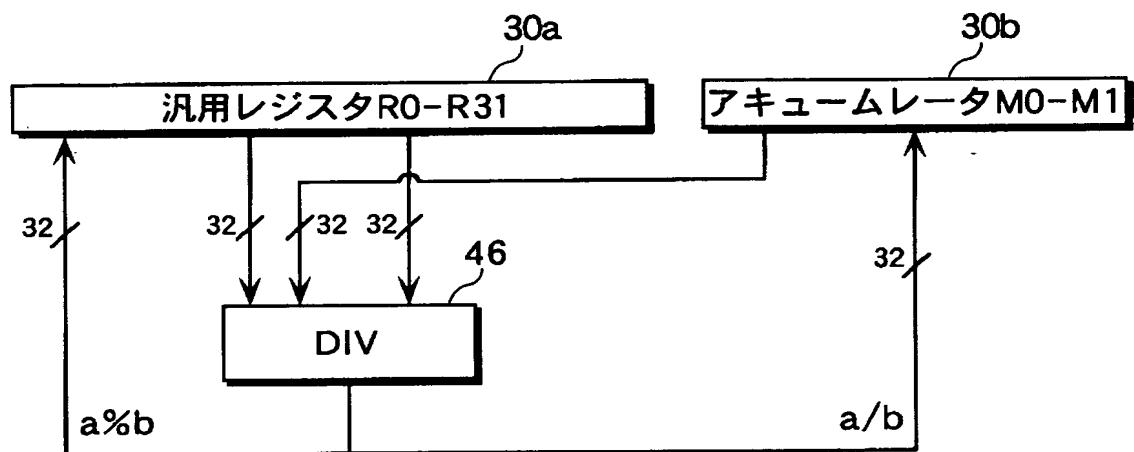
【図3】



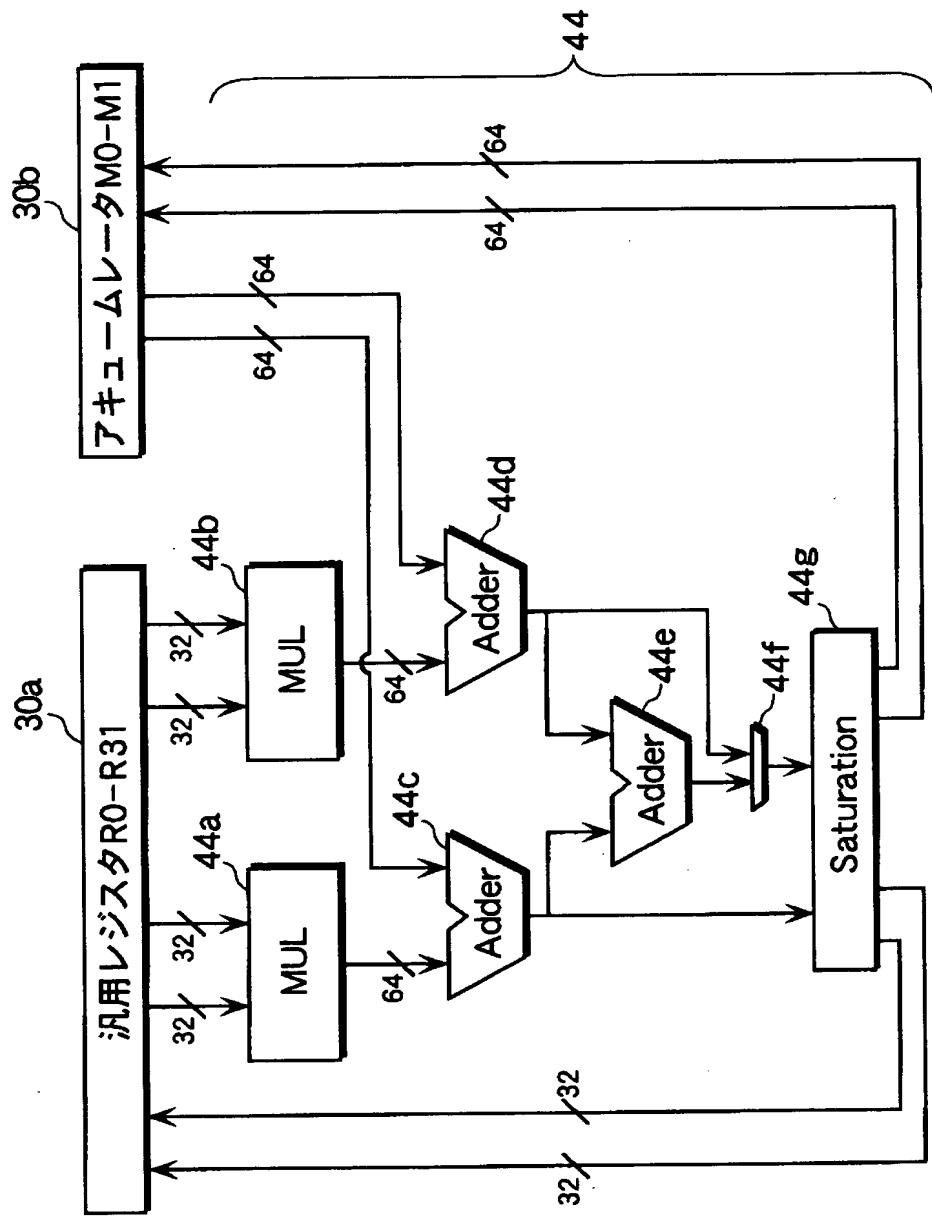
【図4】



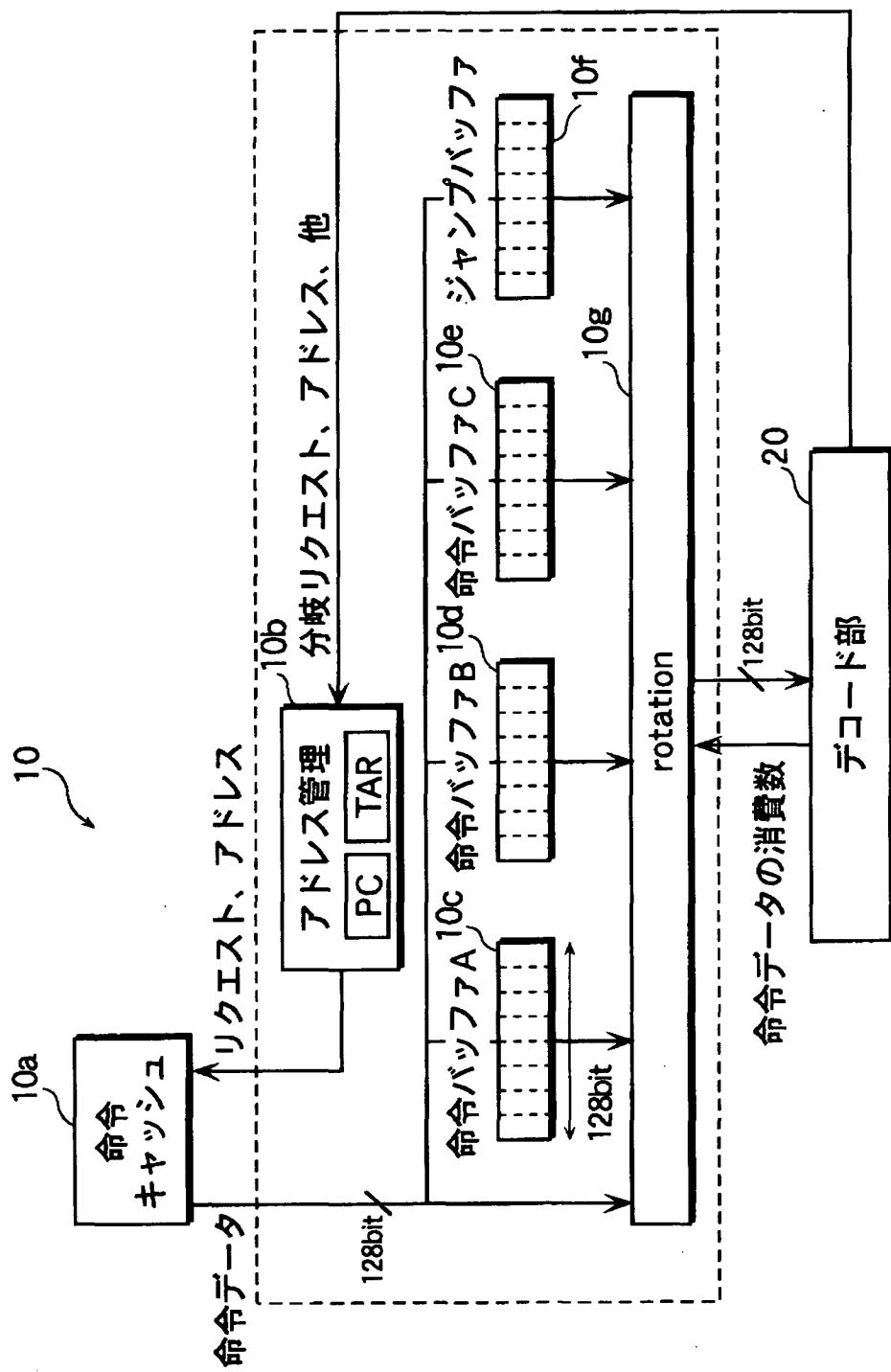
【図5】



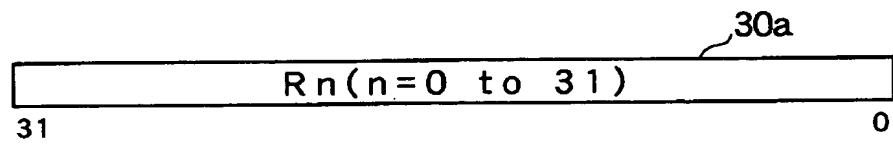
【図6】



【図7】



【図8】



【図9】



【図10】



【図11】

31

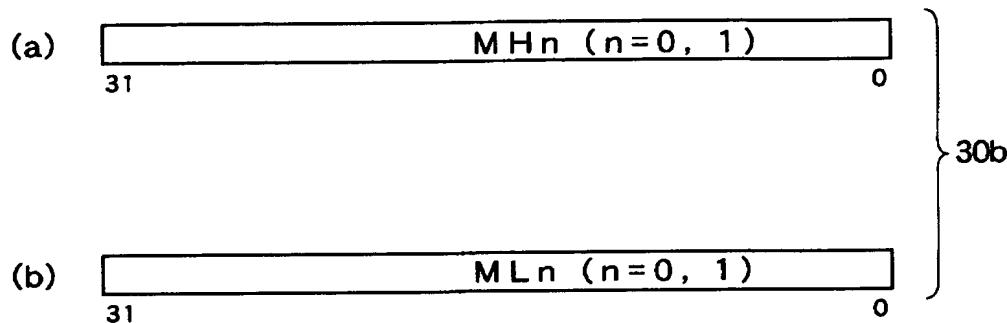
ビット名	31	30	29	28	27	26	25	24
ビット名 reserved	SWE	FXP	reserved	IH	EH		PL	
ビット名	23	22	21	20	19	18	17	16
ビット名 LPIE3	LPIE2	LPIE1	LPIE0	reserved	reserved	AEE	IE	
ビット名	15	14	13	12	11	10	9	8
Reserved								
ビット名	7	6	5	4	3	2	1	0
ビット名	IM[7:0]							

【図12】

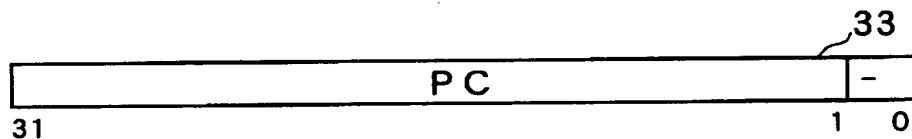
32

ビット名	31	30	29	28	27	26	25	24
ビット名	ALN reserved							
ビット名	23	22	21	20	19	18	17	16
BPO								
ビット名	reserved							
ビット名	15	14	13	12	11	10	9	8
ビット名	reserved							
OVS CAS								
ビット名	7	6	5	4	3	2	1	0
ビット名	C7	C6	C5	C4	C3	C2	C1	C0

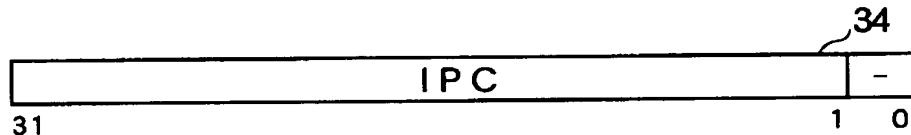
【図13】



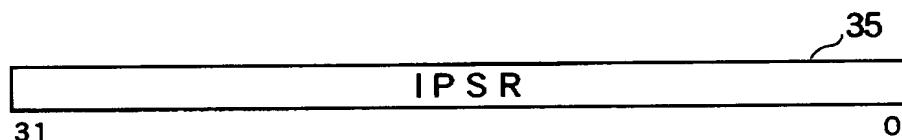
【図14】



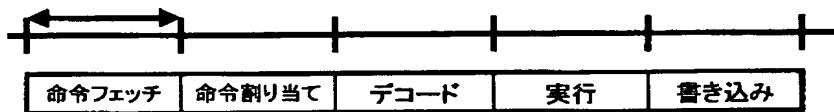
【図15】



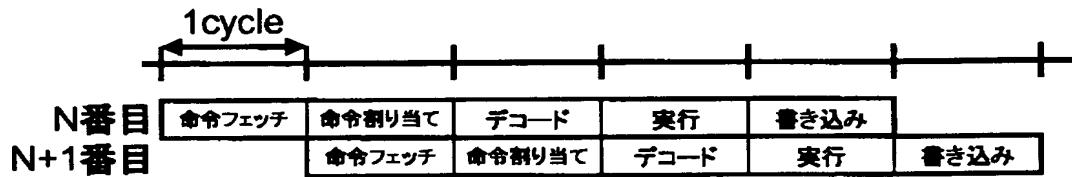
【図16】



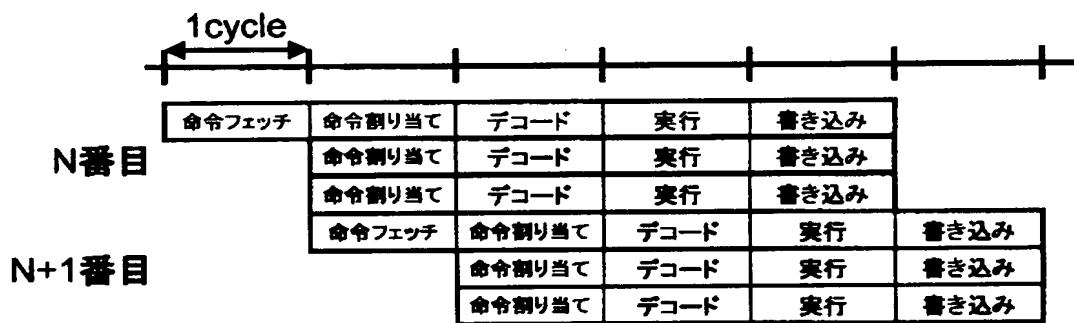
【図17】



【図18】

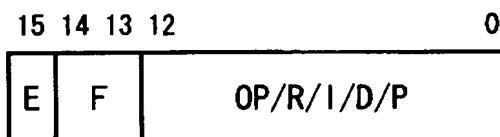


【図19】

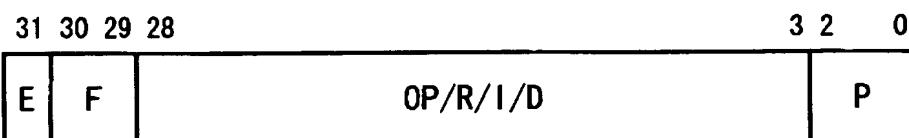


【図20】

(a) 16ビット命令フォーマット



(b) 32ビット命令フォーマット



【図21】

カテゴリ	SIMD サイズ	命令	オペランド	CFR	PSR	代表的な動作	演算器
SINGLE	ワード	add	Rc,Ra,Rb Rb,Ra,I2s SP,I19s ----- Ra2,Rb2 Rc3,Ra3,Rb3 Ra2,I0s SP,I11s				
		addu	Rb,GP,I16u Rb,SP,I16u ----- Ra3,SP,I0u				
		addc	Rc,Ra,Rb W:cas,c0:c1			キャリー付き加算	
		addvw	Rc,Ra,Rb W:ovs			オーバーフローあり加算	
		adds	Rc,Ra,Rb			$Ra + Rb \rightarrow Rc$	32
		addr	Rc,Ra,Rb			$Ra + Rb + 1 \rightarrow Rc$	16
		s1add	Rc,Ra,Rb Rc3,Ra3,Rb3			$Ra + \frac{1}{Rb} \rightarrow Rc$	32
		s2add	Rc,Ra,Rb Rc3,Ra3,Rb3			$Ra + \frac{1}{Rb} \rightarrow Rc$ (>2)	16 32 16
	ハーフワード	addmuk	Rc,Ra,Rb	R:BP0		$Ra [C R B P 0 + ] \rightarrow Rc$	
		addarvw	Rc,Ra,Rb				
ALU add 系	ハーフワード	addvh	Rc,Ra,Rb	W:ovs			
		vaddh	Rc,Ra,Rb			$Ra [16   16] + Rb [16   16] \rightarrow Rc [16   16]$	
		vaddhvh	Rc,Ra,Rb	W:ovs		$Ra [16   16] + Rb [16   16] \rightarrow Rc [16   16]$	
		vsaddh	Rb,Ra,I0s			$Ra [16   16] + 隅値 + 隅値 \rightarrow Rc [16   16]$	
		vaddsh	Rc,Ra,Rb			$Ra [16   16] + Rb [16   16] \rightarrow Rc [16   16]$	
		vaddirh	Rc,Ra,Rb			$(+1) (+1) \rightarrow (丸めあり)$	
		vaddhvc	Rc,Ra,Rb	R:VC			
		varddrhw	Rc,Ra,Rb				
	SIMD	vxaddh	Rc,Ra,Rb			$Ra [16   16] + Rb [16   16] \rightarrow Rc [16   16]$	A 32
		vxaddhv	Rc,Ra,Rb	W:ovs		$Ra [16   16] + Rb [16   16] \rightarrow Rc [16   16]$	
		vhaddh	Rc,Ra,Rb			$Ra [16   16] + Rb [16   16] \rightarrow Rc [16   16]$	
		vhaddhv	Rc,Ra,Rb	W:ovs		$Ra [16   16] + Rb [16   16] \rightarrow Rc [16   16]$	
	バイト	vladdh	Rc,Ra,Rb			$Ra [16   16] + Rb [16   16] \rightarrow Rc [16   16]$	
		vladdhv	Rc,Ra,Rb	W:ovs		$Ra [16   16] + Rb [16   16] \rightarrow Rc [16   16]$	
		vaddb	Rc,Ra,Rb			$Ra [8   8   8   8] + Rb [8   8   8   8] \rightarrow Rc [8   8   8   8]$ (隅値)	
		vsaddb	Rb,Ra,I0s				
		vaddsb	Rc,Ra,Rb			$Ra [8   8   8   8] + Rb [8   8   8   8] \rightarrow Rc [8   8   8   8]$	
		vaddirb	Rc,Ra,Rb			$(+1)(+1)(+1)(+1) \rightarrow (丸めあり)$	

【図22】

カテゴリ	SIMD	サイズ	命令	オペランド	CFR	PSR	代表的な動作	演算器
ALU sub 系	S I N G L E	ワード	sub	Rc,Rb,Rs Rb2,Ra2 Ro3,Rb3,Ra3				31 16
			neub	Rb,Ra;08s Ra2,Rb2 Ra2,04s			即値 - Ra → Rb (Ra2)	32 16
			subc	Rc,Rb,Rs	W:cc,c0xc1		キャリー付き	32 16
			subvw	Rc,Rb,Rs	W:ovs		オーバーフローあり	
			subs	Rc,Rb,Rs			Ra - Ra → 221Ra	
			submask	Rc,Rb,Rs	R:BP0		Ra CFR:BP0 → Rb Rb	
	SIMD	ハーフワード	fsubvh	Rc,Rb,Rs				
			vsubh	Rc,Rb,Rs			Ra [16] [16] - [16] [16] → [16] [16] Rb	
			vsubhvh	Rc,Rb,Rs	W:ovs		Rb [16] [16]	
			vsrauh	Rb,Ra;08s			即値 - 即値 - [16] [16] → [16] [16] Rb	
			vsubsh	Rc,Rb,Rs			Ra [16] [16] - [16] [16] → >>1 [16] [16] Rb	
	SIMD	ハーフワード	vxsubh	Rc,Rb,Rs			Ra [16] [16] - [16] [16] → [16] [16] Rb	
			vxsubhvh	Rc,Rb,Rs	W:ovs		Rb [16] [16]	
			vhsubh	Rc,Rb,Rs			Ra [16] [16] - [16] [16] → [16] [16] Rb	
			vhsubhvh	Rc,Rb,Rs	W:ovs		Rb [16] [16]	
			vlsubh	Rc,Rb,Rs			Ra [16] [16] - [16] [16] → [16] [16] Rb	
	バイト	バイト	vsubb	Rc,Rb,Rs			(即値) Ra [8] [8] [8] [8] - [8] [8] [8] [8] → [8] [8] [8] [8] Rb	
			vsraubb	Rb,Ra;08s				
			vasubb	Rc,Rb,Rs	R:VC			

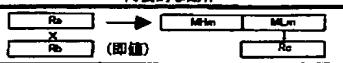
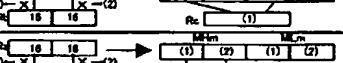
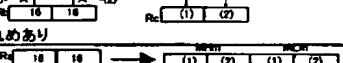
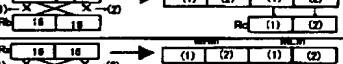
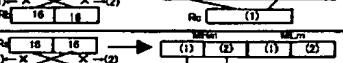
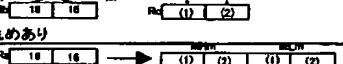
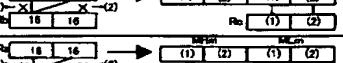
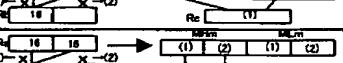
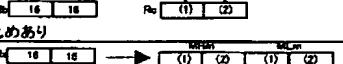
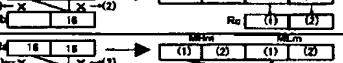
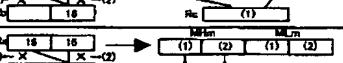
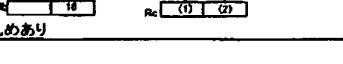
【図23】

カテゴリー	SIMD	サイズ	命令	オペランド	CFR	PSR	代表的な動作	演算器	31 16
ALU logic系	SINGLE	ワード	and	Rc,Ra,Rb Rb,Ra,0Bu Ra2,Rb2			論理積	A	32
			andn	Rc,Ra,Rb Rb,Ra,0Bu Ra2,Rb2					16
			or	Rc,Ra,Rb Rb,Ra,0Bu Ra2,Rb2			論理和		32
			xor	Rc,Ra,Rb Rb,Ra,0Bu Ra2,Rb2			排他的論理和		16
ALU mov系	SINGLE	ワード	mov	Rb,Reg32 Reg32,Rb Rb2,Reg16 Reg16,Rb2 Ra2,Rb Ra16s Ra2A0Bs			Reg32 = TAR LR SVR PSR CFR MH0 MH1 MLO M1 EPSPR IPC IPSR PC EPC PSR0 PSR1 PSR2 PSR3 CFR0 CFR1 CFR2 CFR3 Reg16 = TAR LR MH0 MH1	A	32
			movp	Rc,Rc+1,Ra,Rb			Rc < Ra; Ra+1 < -Rb;		16
			movcf	Ck,Cj,Cm,Cn			Ck < Cj; Cm < Cn;		32
			mvclovs	Cm,Cm+1	W:ovs		Cm:Cm+1 < -CFR.OVS; CFR.OVS; CFR.OVS<-0;		16
			mvclcas	Cm,Cm+1	W:cas		Cm:Cm+1 < -CFR.CAS; CFR.CAS; CFR.CAS<-0;		32
			sethi	Ra16s					32
			max	Rc,Ra,Rb	W:c0.c1		Rc < max(Ra,Rb)		32
ALU max/min系	SIMD	ワード	min	Rc,Ra,Rb	W:c0.c1		Rc < min(Ra,Rb)	A	32
			vmaxh	Rc,Ra,Rb					32
			vminh	Rc,Ra,Rb					32
			バイト	vmaxb	Rc,Ra,Rb				32
			vminb	Rc,Ra,Rb					
ALU abs系	SINGLE		abs	Rb,Ra			絶対値	A	32
			abswv	Rb,Ra	W:ovs		オーバーフローあり		32
			fabsvh	Rb,Ra	W:ovs				32
			vabsvh	Rb,Ra	W:ovs				32
ALU neg系	SINGLE	ワード	negw	Rb,Ra	W:ovs			A	32
			fnegvh	Rb,Ra	W:ovs				32
			vmegvh	Rb,Ra	W:ovs				32
			vmevh	Rb,Ra	W:ovs				32
ALU sum系	SIMD	ハーフワード	vsumh	Rb,Ra			Rb [15 16] + [ ] → Rb [15 16]	A	32
			vsumh2	Rb,Ra			Rb [15 16] [15 16] → Rb [15 16] (1+) (1+) (丸め)		32
			vsumrh2	Rb,Ra					32
			vbesumb	Rc,Ra,Rb					32
ALU その他の	SIMD		frndvh	Rb,Ra	W:ovs		(丸め)	C	32
			vfrndvh	Rb,Mn	W:ovs				32
			vsel	Rc,Ra,Rb	R:VC				32
			vsgnh	Rb,Ra					32

【図24】

カテゴリ	SIMD	サイズ	命令	オペランド	CFR	PSR	代表的な動作	演算器	31 16
CMP	SINGLE		cmpCCn	Cm,Ra,Rb,Cn Cm,Ra,i05s,Cn Cm:Cm+1,Ra,Rb,Cn Cm:Cm+1,Ra,i05s,Cn	W:CF		CC = eq, ne, gt, ge, gtu, geu, le, lt, leu, leu Cm <- result & Cn; (Cm+1 <- ~result & Cn)	A	32
			cmpCCa	Cm:Cm+1,Ra,Rb,Cn Cm:Cm+1,Ra,i05s,Cn	W:CF		Cm <- result & Cn; Cm+1 <- ~(result & Cn);		
			cmpCCo	Cm:Cm+1,Ra,Rb,Cn Cm:Cm+1,Ra,i05s,Cn	W:CF		Cm <- result   Cn; Cm+1 <- ~(result   Cn);		
			cmpCC	C8,Ra2,Rb2 C8,Ra2,i04s	W:CF		CC = eq, ne, gt, ge, le, lt C8 <- result;		
			tstzn	Cm,Ra,Rb,Cn Cm,Ra,i05u,Cn Cm:Cm+1,Ra,Rb,Cn Cm:Cm+1,Ra,i05u,Cn	W:CF		Cm <- (Ra & Rb == 0) & Cn; (Cm+1 <- ~(Ra & Rb == 0) & Cn)		32
			tstza	Cm:Cm+1,Ra,Rb,Cn Cm:Cm+1,Ra,i05u,Cn	W:CF		Cm <- (Ra & Rb == 0) & Cn; Cm+1 <- ~(Ra & Rb == 0) & Cn;		
			tstzo	Cm:Cm+1,Ra,Rb,Cn Cm:Cm+1,Ra,i05u,Cn	W:CF		Cm <- (Ra & Rb == 0)   Cn; Cm+1 <- ~(Ra & Rb == 0)   Cn;		
			tstnn	Cm,Ra,Rb,Cn Cm,Ra,i05u,Cn Cm:Cm+1,Ra,Rb,Cn Cm:Cm+1,Ra,i05u,Cn	W:CF		Cm <- (Ra & Rb != 0) & Cn; (Cm+1 <- ~(Ra & Rb != 0) & Cn)		
			tstna	Cm:Cm+1,Ra,Rb,Cn Cm:Cm+1,Ra,i05u,Cn	W:CF		Cm <- (Ra & Rb != 0) & Cn; Cm+1 <- ~(Ra & Rb != 0) & Cn;		16
			tstno	Cm:Cm+1,Ra,Rb,Cn Cm:Cm+1,Ra,i05u,Cn	W:CF		Cm <- (Ra & Rb != 0)   Cn; Cm+1 <- ~(Ra & Rb != 0)   Cn;		
			tstz	C8,Ra2,Rb2 C8,Ra2,i04u	W:CF		C8 <- (Ra2&Rb2 == 0)		
			tstn	C8,Ra2,Rb2 C8,Ra2,i04u	W:CF		C8 <- (Ra2&Rb2 != 0)		
SIMD	ハーフワード		vcmpCCn	Ra,Rb	W:CF		CC = eq, ne, gt, le, ge, lt	32	
			vscmpCCn	Ra,i05s					
	バイト		vcmpCCb	Ra,Rb	W:CF		CC = eq, ne, gt, le, ge, lt		
			vscmpCCb	Ra,i05s					

【図25】

カテゴリ	SIMD	サイズ	名	オペランド	CFR	PSR	代表的な動作	演算器	31 16	
mul系	S I N G L E	ワード×ワード	mul	Mm,Rc,Ra,Rb Mm,Rb,Ra,I08s				X2	32	
			mulu	Mm,Rc,Ra,Rb Mm,Rb,Ra,I08s			符号なし乗算			
		ワード×ハーフワード	fmuww	Mm,Rc,Ra,Rb		fpx	固定小数点演算	X1		
			hmul	Mm,Rc,Ra,Rb						
		ハーフワード×ハーフワード	lmul	Mm,Rc,Ra,Rb				X1		
			fmulhw	Mm,Rc,Ra,Rb		fpx				
	S I M D	ハーフワード×ハーフワード	fmulhw	Mm,Rc,Ra,Rb		fpx		X2	32	
			fmulhh	Mm,Rc,Ra,Rb		fpx				
		ハーフワード×ワード	fmulhr	Mm,Rc,Ra,Rb		fpx	丸めあり	X2		
			vmul	Mm,Rc,Ra,Rb						
		ハーフワード×ハーフワード	vfmulw	Mm,Rc,Ra,Rb		fpx		X2	32	
			vfmulh	Mm,Rc,Ra,Rb		fpx				
		ワード×ハーフワード	vfmulhr	Mm,Rc,Ra,Rb		fpx	丸めあり	X2		
			vxmul	Mm,Rc,Ra,Rb						
		ハーフワード×ワード	vxfmulw	Mm,Rc,Ra,Rb		fpx		X2	32	
			vxfmulh	Mm,Rc,Ra,Rb		fpx				
		ワード×ハーフワード	vxfmehr	Mm,Rc,Ra,Rb		fpx	丸めあり	X2		
			vhmul	Mm,Rc,Ra,Rb						
		ハーフワード×ハーフワード	vhfmulw	Mm,Rc,Ra,Rb		fpx		X2	32	
			vhfmulh	Mm,Rc,Ra,Rb		fpx				
		ワード×ワード	vhfmehr	Mm,Rc,Ra,Rb		fpx	丸めあり	X2		
			vlmul	Mm,Rc,Ra,Rb						
		ハーフワード×ハーフワード	vfmulw	Mm,Rc,Ra,Rb		fpx		X2	32	
			vfmulh	Mm,Rc,Ra,Rb		fpx				
		ワード×ハーフワード	vfmehr	Mm,Rc,Ra,Rb		fpx	丸めあり	X2		
			vpfmulhw	Mm,Rc,Rc+1,Ra,Rb Mm,Rc,Rc+1,Ra,Rb		fpx				

【図26】

カテゴリ	SIMD	サイズ	命令	オペランド	CFR	PSR	代表的な動作	演算器	演算器
mac系	SINGLE	ワード×ワード	mac	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			mulを使った積和演算	X2	31
			fmacww	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fpx	fmulwwを使った積和演算		
		ワード×ハーフワード	hmac	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			fmulhを使った積和演算	X1	16
			lmac	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			fmulを使った積和演算		
			fmachww	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fpx	fmulhwwを使った積和演算		
			fmachw	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fpx	fmulhwを使った積和演算		
			fmachh	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fpx	fmulhhを使った積和演算		
			fmachhr	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fpx	丸めあり		
	SIMD	ハーフワード×ハーフワード	vmac	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			vmulを使った積和演算	X2	32
			vfmacw	Mm,Rc,Ra,Rb,Mn		fpx	vfmulwを使った積和演算		
			vimac	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			vxmulを使った積和演算		
			vxfmacw	Mm,Rc,Ra,Rb,Mn		fpx	vxfmulwを使った積和演算		
			vximach	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fpx	vximachを使った積和演算		
			vximachr	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fpx	丸めあり		
			vhmac	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			vhmulを使った積和演算		
			vhfmacw	Mm,Rc,Ra,Rb,Mn		fpx	vhfmulwを使った積和演算		
			vhfmach	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fpx	vhfmulhを使った積和演算		
			vhfmachr	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx					
			vimac	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			vimulを使った積和演算		
			vfifmacw	Mm,Rc,Ra,Rb,Mn		fpx	vfifmulwを使った積和演算		
			vfifmach	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fpx	vfifmulhを使った積和演算		
			vfifmachr	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fpx	丸めあり		
			vfifmech	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Mn		fpx	vfifmuhを使った積和演算		
			vfifmechr	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fpx	丸めあり		
		ワード×ハーフワード	vpfmacww	Mm,Rc,Ra,Rb,Mn		fpx			

【図27】

カテゴリ	SIMD	サイズ	命令	オペランド	CFR	PSR	代表的な動作	演算器	31 16	
SINGLE	msu	ワード×ワード	msu	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			mulを使った積差演算	X2	32	
			fmsuww	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	fmulwwを使った積差演算			
		ワード ×ハーフワード	hmsu	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			hmulを使った積差演算	X1		
			lmsu	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			lmulを使った積差演算			
		ハーフワード ×ハーフワード	fmsuhw	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	fmulthwを使った積差演算	X2		
			fmsuhh	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	fmulthhを使った積差演算			
	vmsu	ハーフワード ×	vmsu	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			vmulを使った積差演算	X2	32	
			vfmsuw	Mm,Rc,Ra,Rb,Mn		fxp	vfmulを使った積差演算			
			vfmsuh	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	vfmult使用的積差演算			
			vxmsu	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			vxmulを使った積差演算			
			vxvfmsuw	Mm,Rc,Ra,Rb,Mn		fxp	vxvfmultwを使った積差演算			
			vxvfmsuh	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	vxvfmulthを使った積差演算			
		ハーフワード	vhmsu	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			vhmulを使った積差演算	X2	32	
			vhvfmsuw	Mm,Rc,Ra,Rb,Mn		fxp	vhvfmultwを使った積差演算			
			vhvfmsuh	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	vhvfmulthを使った積差演算			
			vlmsu	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx			vlmulを使った積差演算			
			vlvfmsuw	Mm,Rc,Ra,Rb,Mn		fxp	vlvfmultwを使った積差演算			
			vlvfmsuh	Mm,Rc,Ra,Rb,Mn M0,Rc,Ra,Rb,Rx		fxp	vlvfmulthを使った積差演算			

【图28】

カテゴリ	SIMD サイズ	命令	オペランド	CFR	PSR	代表的な動作	演算器
MEM Id 系	ワード	ld	Rb,(Ra,d10u) Rb,(GP,d13u) Rb,(SP,d13u) Rb,(Ra+d10s) Rb2,(Ra2) Rb2,(Ra2,d05u) Rb2,(GP,d06u) Rb2,(SP,d06u) Rb2,(Ra2+)			レジスタ 32 ← メモリ 32	31 16
		ldh	Rb,(Ra,d09u) Rb,(GP,d12u) Rb,(SP,d12u) Rb,(Ra+d09s) Rb2,(Ra2) Rb3,(Ra2,d04u) Rb2,(GP,d05u) Rb2,(SP,d05u) Rb2,(Ra2+)			32 ← 16	32
		ldhu	Rb,(Ra,d09u) Rb,(GP,d12u) Rb,(SP,d12u) Rb,(Ra+d09s)				16
		ldb	Rb,(Ra,d08u) Rb,(GP,d11u) Rb,(SP,d11u) Rb,(Ra+d08s)			レジスタ 32 ← メモリ 8	M 32
		ldbu	Rb,(Ra,d08u) Rb,(GP,d11u) Rb,(SP,d11u) Rb,(Ra+d08s)				32
	シングル	バイト→バイト	ldbh	Rb,(Ra+d07s)		16 ← 8	32
		ハーフワード	ldbh	Rb,(Ra+d07s)		16 ← 8	32
	ペア	ワード	Rb;Rb+1,(Ra,d11u) LR;SVR,(Ra,d11u) TAR;UDR,(Ra,d11u) Rb;Rb+1,(GP,d14u) LR;SVR,(GP,d14u) TAR;UDR,(GP,d14u) Rb;Rb+1,(SP,d14u) LR;SVR,(SP,d14u) TAR;UDR,(SP,d14u) Rb;Rb+1,(Ra+d11s) Rb;Rb+1,(SP,d07u) LR;SVR,(SP,d07u) Rb2;Rb2,(Ra2+)			32 ← 32	32
		ハーフワード	ldhp	Rb;Rb+1,(Ra,d10u) Rb;Rb+1,(Ra+d10s) Rb2;Rb2,(Ra2+)		32 ← 16 16	32
		バイト	ldbp	Rb;Rb+1,(Ra,d09u) Rb;Rb+1,(Ra+d09s)		32 ← 8 8	16
		バイト→ハーフワード	ldbhp	Rb;Rb+1,(Ra+d07s)		16 ← 8 8 8	32
			ldbuhp	Rb;Rb+1,(Ra+d07s)		16 ← 8 8 8	32

【図29】

カテゴリ	SIMD	サイズ	命令	オペランド	CFR	PSR	代表的な動作	演算器	31 16
SINGLE MEM store系	S	ワード	st	(Ra,d10u).Rb (GP,d13u).Rb (SP,d13u).Rb (Ra+d10s).Rb (Ra2).Rb2 (Ra2,d05u).Rb2 (GP,d06u).Rb2 (SP,d06u).Rb2 (Ra2+).Rb2			レジスタ 32 → メモリ 32		32
			sth	(Ra,d09u).Rb (GP,d12u).Rb (SP,d12u).Rb (Ra+d09s).Rb (Ra2).Rb2 (Ra2,d04u).Rb2 (GP,d05u).Rb2 (SP,d05u).Rb2 (Ra2+).Rb2			16 → 16		16
		ハーフワード	stb	(Ra,d08u).Rb (GP,d11u).Rb (SP,d11u).Rb (Ra+d08s).Rb			8 → 8		32
			stbh	(Ra+d07s).Rb			1 8 → 16		16
	P	ペア	stp	(Ra,d11u).Rb:Rb+1 (Ra,d11u).LR:SVR (Ra,d11u).TAR:UDR (GP,d14u).Rb:Rb+1 (GP,d14u).LRSVR (GP,d14u).TAR:UDR (SP,d14u).Rb:Rb+1 (SP,d14u).LR:SVR (SP,d14u).TAR:UDR (Ra+d11s).Rb:Rb+1 (SP,d07u).Rb:Rb (SP,d07u).LR:SVR (Ra2+).Rb2:Rb2			32 → 32	M	32
			sthp	(Ra,d10u).Rb:Rb+1 (Ra+d10s).Rb:Rb+1 (Ra2+).Rb2:Rb2			16 → 32		16
			stbp	(Ra,d09u).Rb:Rb+1 (Ra+d09s).Rb:Rb+1			16 → 16		32
		ペア→ハーフワード	stbhp	(Ra+d07s).Rb:Rb+1			1 1 8 → 32		16

【図30】

カテゴリ	SIMD	サイズ	命令	オペランド	CFR	PSR	代表的な動作	演算器	31 16
BRA			setlr	d09s C5,d09s			LRを設定 LRからフェッチした命令を分岐用バッファに格納	B	32
			settar	d09s C6,d09s C6,C2:C4,d09s C6,Cm,d09s C6,C4,d09s	W:c6 W:c2:c4,c6 W:c6.cm W:c6		TARを設定 TARからフェッチした命令を分岐用バッファに格納		16
			setbb	LR TAR			LRからフェッチした命令を分岐用バッファに格納 TARからフェッチした命令を分岐用バッファに格納		32
			jloop	C5,LR,Ra,j08s C6,TAR,Ra,j08s C6,C2:C4,TAR,Ra,j08s C6,Cm,TAR,Ra,j08s C6,TAR,Ra2 C6,C2:C4,TAR,Ra2 C6,Cm,TAR,Ra2	W:c5 W:c6 W:c2:c4,c6 W:c6.cm W:c6 W:c2:c4,c6 W:c6		ブレディケート[c5]のみ ブレディケート[c6]のみ		16
			jmp	TAR LR					32
			jmpf	TAR LR Cm,TAR C6,C2:C4,TAR	R:CF				16
			jmpo	LR					32
			br	d20s d09s			ブレディケート[c6][c7]のみ		16
			bri	d20s d09s	R:CF				32
			rti			W:PSR Reh			16

【図31】

カテゴリ	SIMD	サイズ	命令	オペランド	CFR	PSR	代表的な動作	演算器	31 16
BS asl 系	S I N G L E	ワード	asl	Rc,Ra,Rb Rb,Ra,I05u Ra2,I04u			左シフト <<Ra(0:4) Ra → Rc	S1	32 16
			fashvw	Rc,Ra,Rb Rb,Ra,I05u Rc,Ra,Rb Rb,Ra,I05u	W:ovs		<<Ra(0:4) Ra → Rc 諸和あり		
		ペアワード	aslp	Mm,Ra,Mn,Rb Mm,Rb,Mn,I06u Mm,Rc,MHn,Ra,Rb Mm,Rb,MHn,Ra,I06u			<<Ra(0:4) Ml_m Mn_m → Ml_m Mn_m Ra	S2	32
			fastpvw	Mm,Ra,Mn,Rb Mm,Rb,Mn,I06u	W:ovs		<<Ra(0:4) Ml_m Mn_m → Ml_m Mn_m Ra		
			vasl	Mm,Ra,Mn,Rb Mm,Rb,Mn,I05u			Ml_m Mn_m (1) (2) (1) (2) → (1) (2) (1) (2) (1)<<Ra(0:4) (2)<<Ra(0:4) Ra		
	S I M D	ワード	vfashvw	Mm,Ra,Mn,Rb Mm,Rb,Mn,I05u	W:ovs		Ml_m Mn_m (1) (2) (1) (2) → (1) (2) (1) (2) (1)<<Ra(0:4) (2)<<Ra(0:4) Ra	S2	32
			vaslh	Rc,Ra,Rb Rb,Ra,I04u			<<Ra(0:2) <<Ra(0:2) Ra (1) (2) → (1) (2) Ra		
		ハーフワード	vfashvh	Rc,Ra,Rb Rb,Ra,I04u	W:ovs		<<Ra(0:2) <<Ra(0:2) Ra (1) (2) → (1) (2) Ra 諸和あり	S1	32
			vaslb	Rc,Ra,Rb Rb,Ra,I03u			<<Ra(0:2) <<Ra(0:2) Ra (1) (2) (3) (4) → (1) (2) (3) (4) Ra <<Ra(0:2) <<Ra(0:2)		
		BS asr 系	asr	Rc,Ra,Rb Rb,Ra,I05u Ra2,I04u			算術右シフト >>Ra(0:4) Ra → Rc	S1	32 16
			assrp	Mm,Ra,Mn,Rb Mm,Rb,Mn,I06u Mm,Rc,MHn,Ra,Rb Mm,Rb,MHn,Ra,I06u			>>Ra(0:4) Ml_m Mn_m → Ml_m Mn_m Ra		
		ワード	vaser	Mm,Ra,Mn,Rb Mm,Rb,Mn,I05u			Ml_m Mn_m (1) (2) (1) (2) → (1) (2) (1) (2) (1)>>Ra(0:4) (2)>>Ra(0:4) Ra	S2	32
			vasrh	Rc,Ra,Rb Rb,Ra,I04u			>>Ra(0:2) >>Ra(0:2) Ra (1) (2) → (1) (2) Ra		
			vasrb	Rc,Ra,Rb Rb,Ra,I03u			>>Ra(0:2) >>Ra(0:2) Ra (1) (2) (3) (4) → (1) (2) (3) (4) Ra >>Ra(0:2) >>Ra(0:2)		

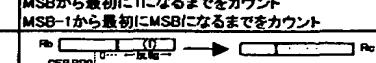
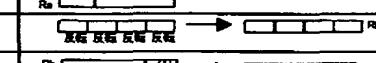
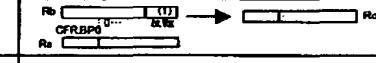
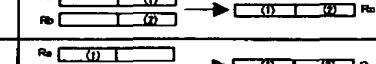
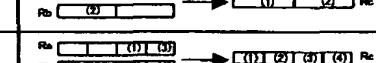
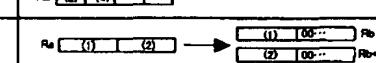
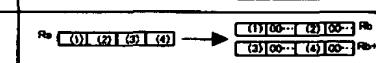
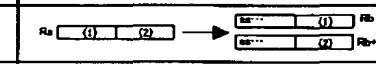
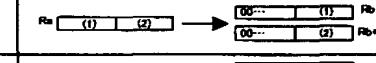
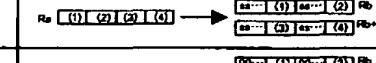
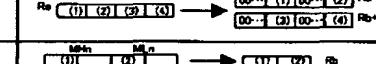
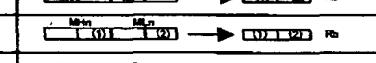
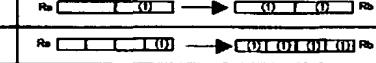
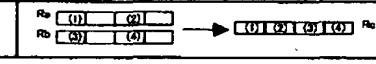
【図32】

カテゴリ	SIMD	サイズ	命令	オペランド	CFR	PSR	代表的な動作	演算器	31 16
BS ler系	SINGLE	ワード	lsr	Rc,Ra,Rb Rb,Ra,i05u			論理右シフト >>Ra(0:4) Ra → Ra	S1	32
		ペアワード	lsep	Mm,Ra,Mn,Rb Mm,Rb,Mn,i06u Mm,Rc,MHn,Ra,Rb Mm,Rb,MHn,Ra,i06u			>>Ra(0:4) Mm,Mn → Mm,Mn Ra → Ra		
	SIMD	ワード	vlsr	Mm,Ra,Mn,Rb Mm,Rb,Mn,i05u			Mm,Mn (1)(2) (1)(2) → (1)(2) (1)(2) (1)>>Ra(0:4) (2)>>Ra(0:4) Ra → Ra	S2	32
		ハーフワード	vlsh	Rc,Ra,Rb Rb,Ra,i04u			>>Ra(0:3) >>Ra(0:3) Ra (1)(2) → (1)(2) Ra	S1	
	SIMD	バイト	vlshb	Rc,Ra,Rb Rb,Ra,i03u			>>Ra(0:2) >>Ra(0:2) Ra (1)(2)(3)(4) → (1)(2)(3)(4) Ra >>Ra(0:2) >>Ra(0:2)	S1	
	SINGLE	ワード	rol	Rc,Ra,Rb Rb,Ra,i05u			Ra <<Ra(4:0) <<Ra(4:0) → Ra	S1	32
	SIMD	ハーフワード	vrolh	Rc,Ra,Rb Rb,Ra,i04u				S1	
		バイト	vrolb	Rc,Ra,Rb Rb,Ra,i03u				S1	
BS ext系	SINGLE	ワード	extw	Mm,Rb,Ra			Ra → Mm Mm Ra Ra → Ra	C	32
		ハーフワード	exth	Ra2			Ra4 (0) → (0) Ra2	S2	16
			exthu	Ra2			Ra4 (0) → (0) Ra2		
		バイト	extb	Ra2			Ra4 (0) → (0) Ra2		
	SIMD	ハーフワード	extbu	Ra2			Ra4 (0) → (0) Ra2	C	32

【図33】

カテゴリ	SIMD	サイズ	命令	オペランド	CFR	PSR	代表的な動作	演算器	31 16
CNV valn系	SIMD		valn	Rc,Ra,Rb	Rain[1:0]		<<(OPR1,MN[1:0]<<3) [B3:32] Ra Ra → Ra	C	32
			valn1	Rc,Ra,Rb			Ra (0) → (0) Ra		
			valn2	Rc,Ra,Rb			Ra (0) → (0) Ra		
			valn3	Rc,Ra,Rb			Ra (0) → (0) Ra		
			valnvcl	Rc,Ra,Rb	RVC0				
			valnvcl	Rc,Ra,Rb	RVC0				
			valnvcl	Rc,Ra,Rb	RVC0				
			valnvcl	Rc,Ra,Rb	RVC0				

【図34】

カテ ゴリ	SIMD	サイズ	命令	オペランド	CFR	PSR	代表的な動作	演算 器	31 16
CNV	S I N G L E		bentl	Rb,Ra			1の数をカウント	C	31 16
			bseq0	Rb,Ra			MSBから最初に0になるまでをカウント		
			bseq1	Rb,Ra			MSBから最初に1になるまでをカウント		
			bsaq	Rb,Ra			MSB-1から最初にMSBになるまでをカウント		
			mskbrvh	Rc,Ra,Rb	R:BP0				
	SIMD	ハーフワード	byterev	Rb,Ra				C	32
			mskbrvb	Rc,Ra,Rb	R:BP0				
		バイト	virtlh	Rc,Ra,Rb					
			virtlhh	Rc,Ra,Rb					
			virtlib	Rc,Ra,Rb					
		SIMD	virtlhb	Rc,Ra,Rb					
			vhumplh	Rb:Rb+1,Ra					
			vhumplb	Rb:Rb+1,Ra					
			vlumpkh	Rb:Rb+1,Ra					
			vkunkphu	Rb:Rb+1,Ra					
		バイト	vlumpkb	Rb:Rb+1,Ra					
			vkunkpku	Rb:Rb+1,Ra					
			vunpk1	Rb,Mn					
		ハーフワード	vunpk2	Rb,Mn					
			vstovh	Rb,Ra					
			vstovb	Rb,Ra					
			vhpkb	Rc,Ra,Rb					

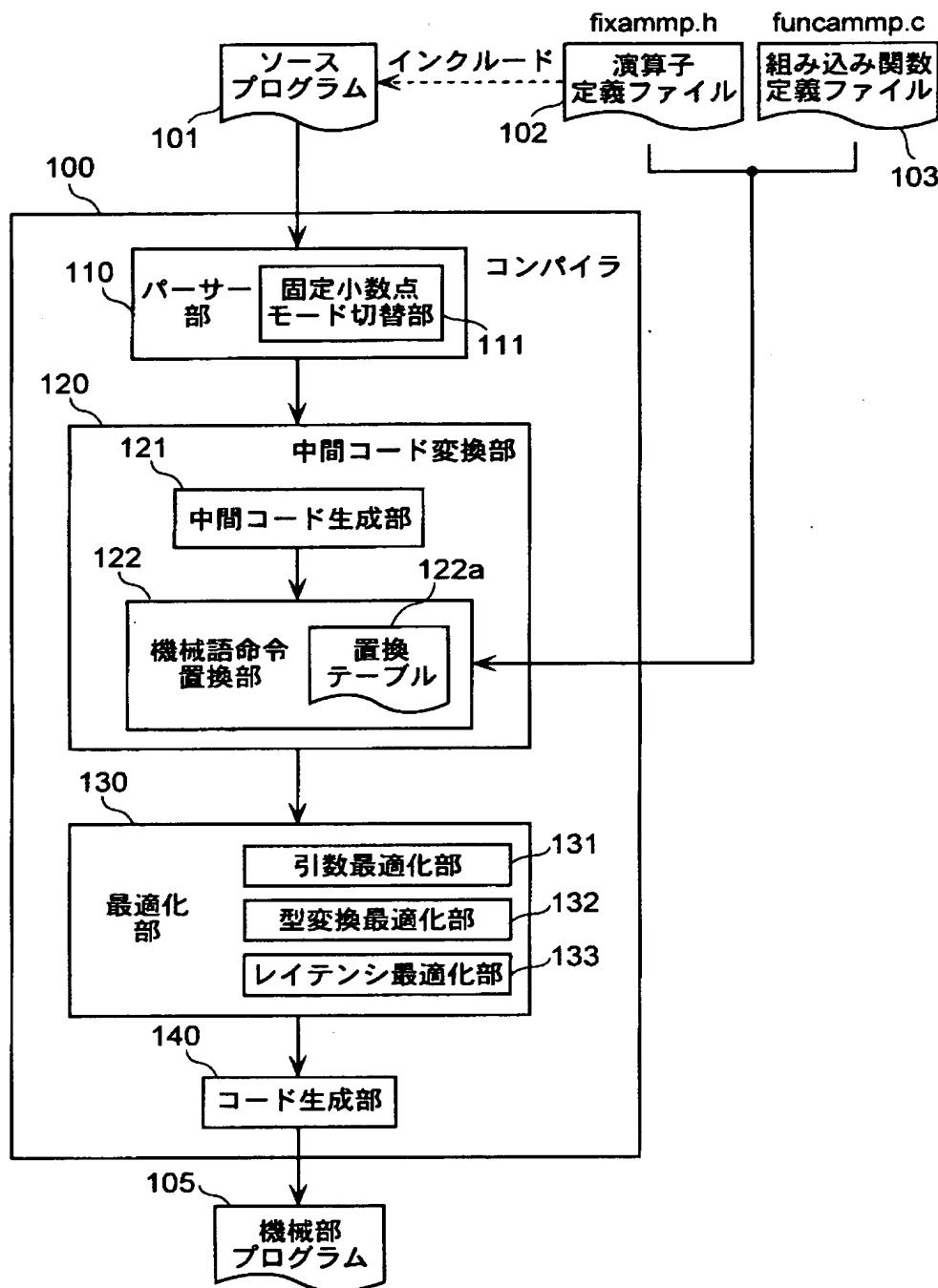
【図35】

カテゴリ	SIMD	サイズ	命令	オペランド	CFR	PSR	代表的な動作	演算器	31 16
SAT vpk系	SIMD	ワード→ハーフワード	vpkh	Rc,Ra,Rb			Rb (1) 鈍和あり Rb (2) → (1) (2) Rb	C	32
			vpkhu	Rc,Ra,Rb			Rb (1) 符号なし Rb (2) 鈍和あり → (1) (2) Rb		
	SIMD	ハーフワード→バイト	vpkb	Rc,Ra,Rb			Rb (1) (2) 鈍和あり Rb (3) (4) → (1) (2) (3) (4) Rb	C	32
			vpkbu	Rc,Ra,Rb			Rb (1) (2) 符号なし Rb (3) (4) 鈍和あり → (1) (2) (3) (4) Rb		
SAT sat系	SINGLE	ワード	setw	Mm,Rb,Mn			ワード鈍和	C	32
			seth	Rb,Ra			ハーフワード鈍和		
			setb	Rb,Ra			バイト鈍和		
			setbu	Rb,Ra			バイト符号なし鈍和		
			set9	Rb,Ra			9ビット鈍和		
			set12	Rb,Ra			12ビット鈍和		
	SIMD	ハーフワード	vsath	Mm,Rb,Mn			Rb (1) (2) Mm (1) (2) 鈍和あり Mm (1) (2) Rb	C	32
			vsath8	Rb,Ra			符号付き8ビット鈍和		
			vsath8u	Rb,Ra			符号なし8ビット鈍和		
			vsath9	Rb,Ra			9ビット鈍和		
			vsath12	Rb,Ra			12ビット鈍和		

【図36】

カテゴリ	SIMD	サイズ	命令	オペランド	CFR	PSR	代表的な動作	演算器	31 16
MSK			mskgen	Rc,Rb Rb,I05U,I05u			マスク生成 Rb(128) Rb(40) or Rb (<0) Rb(128) 0... (0...) Rb 1... 0... 1... Rb(128) Rb(40) Rb(40) Rb(128)	S2	32
			msk	Rc,Ra,Rb Rb,Ra,I05U,I05u			Rb (1) (2) Rb (1) (2) Rb (1) (2) Rb (1) (2)		
EXTR			extr	Rc,Ra,Rb Rb,Ra,I05U,I05u			Rb(128) Rb(40) 符号拡張あり (符号拡張なし)	S2	32
			extru	Rc,Ra,Rb Rb,Ra,I05U,I05u			Rb (1) (2) Rb (1) (2) Rb (1) (2)		
DIV			div	Mm,Rc,Mm,Ra,Rb Mm,Rc,Mm,Ra,Rb	W:ows		除算	DIV	32
ETC			piNl				Wih,ie,pe R:PSR ソフトウェア割り込みN=0~7	B	32
			piN				Wih,ie,pl R:PSR ソフトウェア割り込みN=0~7		
			scN				Wih,ie,pl R:PSR システムコールN=0~7		
			ldstb	Rb,(Ra)			load/バスロック	M	32
			rd	Rb,(Ra) Rb,(d1)u Rb2,(Ra2)		R:ee	外部レジスタリード		
			wt	(Ra),Rb (d1)u,Rb (Ra2),Rb2		R:ee	外部レジスタライト	DBGM	16
			dpref	(Ra,d1)u			ブリフェッチ		
			dbgmN	i18u			N=0~3		
			vccnk		W:CF R:VC		VCフラグチェック	B	32
			vmpaw				VMP切替え		
			vmpnew	LR			no operation		
			vmpintd1			W:ie	VMP切替え禁止		
			vmpintd2			W:ie	VMP切替え許可	A	16
			vmpintd3						
			vmpinte1						
			vmpinte2						
			vmpinte3						
			nop						

【図37】



【図38】

```

/*
 * (C)Copyright 2002 Matsushita Electric Industrial Co., Ltd.
 * fixammp.h
 * Version:
 * Release:
 * Date: 2002/6/14 v0.9.1 対応
 *
 ****
 */

/* 多重定義回避 */
#ifndef __FIXAMMP__
#define __FIXAMMP__

/* FIX-Lib. クラス定義 */
class FIX16_1;
class FIX32_1;
class FIX16_2;
class FIX32_2;

#if defined(__AMMPCC__)
#pragma pack_struct
#endif // __AMMPCC__

////////////////////////////// Member of FIX16_1 ///////////////////
//
// val : 実際の 16bit の値
//
////////////////////////////// FIX16_1 ///////////////////
class FIX16_1{
    short val;
public:
    // constructor
    FIX16_1(){}
    FIX16_1(int a);
    FIX16_1(float a);
    FIX16_1(double a);
    FIX16_1(FIX16_1& a)
    {
        val = a.val;
    }
    FIX16_1(volatile FIX16_1& a)
    {
        val = a.val;
    }
    FIX16_1(const FIX16_1& a)
    {
        val = a.val;
    }

    // Operator
    volatile FIX16_1& operator=(FIX16_1 a) volatile
    {
        val = a.val;
        return *this;
    }
    FIX16_1& operator=(FIX16_1 a)
    {

```

【図39】

```

    val = a.val;
    return *this;
}
friend FIX16_1 operator+(FIX16_1 a);
friend FIX16_1 operator-(FIX16_1 a);

friend FIX16_1 operator+(FIX16_1 a, FIX16_1 b);
friend FIX16_1 operator-(FIX16_1 a, FIX16_1 b);

friend FIX16_1 operator*(FIX16_1 a, FIX16_1 b);
friend FIX16_1 operator*(int a, FIX16_1 b);
friend FIX16_1 operator*(FIX16_1 a, int b);
friend FIX16_1 operator*(float a, FIX16_1 b);
friend FIX16_1 operator*(FIX16_1 a, float b);
friend FIX16_1 operator*(double a, FIX16_1 b);
friend FIX16_1 operator*(FIX16_1 a, double b);

friend FIX16_1 operator/(FIX16_1 a, FIX16_1 b);
friend FIX16_1 operator/(FIX16_1 a, int b);
friend FIX16_1 operator/(FIX16_1 a, float b);
friend FIX16_1 operator/(FIX16_1 a, double b);

friend FIX16_1 operator<<(FIX16_1 a, int b);
friend FIX16_1 operator>>(FIX16_1 a, int b);

friend bool operator<(FIX16_1 a, FIX16_1 b);
friend bool operator>(FIX16_1 a, FIX16_1 b);
friend bool operator<=(FIX16_1 a, FIX16_1 b);
friend bool operator>=(FIX16_1 a, FIX16_1 b);
friend bool operator==(FIX16_1 a, FIX16_1 b);
friend bool operator!=(FIX16_1 a, FIX16_1 b);

volatile FIX16_1& operator<=(int b) volatile;
    FIX16_1& operator<=(int b);
volatile FIX16_1& operator>=(int b) volatile;
    FIX16_1& operator>=(int b);

volatile FIX16_1& operator*=(FIX16_1 b) volatile;
    FIX16_1& operator*=(FIX16_1 b);
volatile FIX16_1& operator*=(int b) volatile;
    FIX16_1& operator*=(int b);
volatile FIX16_1& operator*=(float b) volatile;
    FIX16_1& operator*=(float b);
volatile FIX16_1& operator*=(double b) volatile;
    FIX16_1& operator*=(double b);

volatile FIX16_1& operator/=(FIX16_1 b) volatile;
    FIX16_1& operator/=(FIX16_1 b);
volatile FIX16_1& operator/=(int b) volatile;
    FIX16_1& operator/=(int b);
volatile FIX16_1& operator/=(float b) volatile;
    FIX16_1& operator/=(float b);
volatile FIX16_1& operator/=(double b) volatile;
    FIX16_1& operator/=(double b);
volatile FIX16_1& operator+=(FIX16_1 b) volatile;
    FIX16_1& operator+=(FIX16_1 b);
volatile FIX16_1& operator-=(FIX16_1 b) volatile;
    FIX16_1& operator-=(FIX16_1 b);

```

【図40】

```

short value(){return val;}

// Other functions

friend FIX16_1 _fix161(short a);
friend short _bptn(FIX16_1 a);
friend FIX16_1 fix161(FIX32_1 a);
friend float _fToat(FIX16_1 a);
friend double _double(FIX16_1 a);

friend FIX16_1 _abs(FIX16_1 a);
friend FIX16_1 _max(FIX16_1 a, FIX16_1 b);
friend FIX16_1 _min(FIX16_1 a, FIX16_1 b);
friend FIX16_1 _adds(FIX16_1 a, FIX16_1 b);
friend FIX16_1 _subs(FIX16_1 a, FIX16_1 b);
friend int _bcnt1(FIX16_1 a);
friend int _bseq(FIX16_1 a);
friend int _bseq0(FIX16_1 a);
friend int _bseq1(FIX16_1 a);
friend FIX16_1 round(FIX32_1 a);
friend int _extr(FIX16_1 a, unsigned int b, unsigned int c);
friend unsigned int _extru(FIX16_1 a, unsigned int b, unsigned int c);

friend void _mulr(FIX16_1 &c, FIX16_1 a, FIX16_1 b);
friend void _mulr(long &mh, long&ml, FIX16_1 &c, FIX16_1 a, FIX16_1 b);
friend void _mul(long &mh, long&ml, FIX16_1 &c, FIX16_1 a, FIX16_1 b);
friend void _mul(long &mh, long&ml, FIX32_1 &c, FIX16_1 a, FIX16_1 b);
friend void _mul(long &mh, long&ml, FIX32_1 &c, FIX16_1 a, FIX32_1 b);
friend void _mul(long &mh, long&ml, FIX32_1 &c, FIX32_1 a, FIX16_1 b);
friend void _mac(long &mh, long &ml, FIX16_1 &c, FIX16_1 a, FIX16_1 b);
friend void _mac(long &mh, long &ml, FIX32_1 &c, FIX16_1 a, FIX16_1 b);
friend void _mac(long &mh, long &ml, FIX32_1 &c, FIX32_1 a, FIX16_1 b);
friend void _macr(long &mh, long &ml, FIX16_1 &c, FIX16_1 a, FIX16_1 b);
friend void _msu(long &mh, long&ml, FIX16_1 &c, FIX16_1 a, FIX16_1 b);
friend void _msu(long &mh, long&ml, FIX32_1 &c, FIX16_1 a, FIX16_1 b);
friend void _msu(long &mh, long&ml, FIX32_1 &c, FIX32_1 a, FIX16_1 b);
friend void _msur(long &mh, long&ml, FIX16_1 &c, FIX16_1 a, FIX16_1 b);
friend FIX16_1 _div(FIX16_1 a, FIX16_1 b);
friend FIX16_1 _div(FIX16_1 a, int b);
friend FIX16_1 _div(FIX16_1 a, float b);
friend FIX16_1 _div(FIX16_1 a, double b);

};

////////////////////////////////////////////////////////////////
// Member of FIX32_1
//
//          val : 実際の32bitの値
//
class FIX32_1{
    long val;
public:
    // constructor
    FIX32_1(){};
    FIX32_1(int a);
}

```

【図4-1】

```

FIX32_1(float a);
FIX32_1(double a);
FIX32_1(FIX16_1 a);
FIX32_1(FIX32_1& a)
{
    val = a.val;
}
FIX32_1(volatile FIX32_1& a)
{
    val = a.val;
}
FIX32_1(const FIX32_1& a)
{
    val = a.val;
}

// Operator
volatile FIX32_1& operator=(FIX32_1 a) volatile
{
    val = a.val;
    return *this;
}
FIX32_1& operator=(FIX32_1 a)
{
    val = a.val;
    return *this;
}
friend FIX32_1 operator+(FIX32_1 a);
friend FIX32_1 operator-(FIX32_1 a);

friend FIX32_1 operator+(FIX32_1 a, FIX32_1 b);
friend FIX32_1 operator-(FIX32_1 a, FIX32_1 b);

friend FIX32_1 operator*(FIX32_1 a, FIX32_1 b);
friend FIX32_1 operator*(int a, FIX32_1 b);
friend FIX32_1 operator*(FIX32_1 a, int b);
friend FIX32_1 operator*(float a, FIX32_1 b);
friend FIX32_1 operator*(FIX32_1 a, float b);
friend FIX32_1 operator*(double a, FIX32_1 b);
friend FIX32_1 operator*(FIX32_1 a, double b);

friend FIX32_1 operator/(FIX32_1 a, FIX32_1 b);
friend FIX32_1 operator/(FIX32_1 a, int b);
friend FIX32_1 operator/(FIX32_1 a, float b);
friend FIX32_1 operator/(FIX32_1 a, double b);

friend FIX32_1 operator<<(FIX32_1 a, int b);
friend FIX32_1 operator>>(FIX32_1 a, int b);

friend bool operator<(FIX32_1 a, FIX32_1 b);
friend bool operator>(FIX32_1 a, FIX32_1 b);
friend bool operator<=(FIX32_1 a, FIX32_1 b);
friend bool operator>=(FIX32_1 a, FIX32_1 b);
friend bool operator==(FIX32_1 a, FIX32_1 b);
friend bool operator!=(FIX32_1 a, FIX32_1 b);

volatile FIX32_1& operator<=(int b) volatile;
    FIX32_1& operator<=(int b);
volatile FIX32_1& operator>=(int b) volatile;

```

【図42】

```

    FIX32_1& operator>>=(int b);

volatile FIX32_1& operator*=(FIX32_1 b) volatile;
FIX32_1& operator*=(FIX32_1 b);
volatile FIX32_1& operator*=(int b) volatile;
FIX32_1& operator*=(int b);
volatile FIX32_1& operator*=(float b) volatile;
FIX32_1& operator*=(float b);
volatile FIX32_1& operator*=(double b) volatile;
FIX32_1& operator*=(double b);

volatile FIX32_1& operator/=(FIX32_1 b) volatile;
FIX32_1& operator/=(FIX32_1 b);
volatile FIX32_1& operator/=(int b) volatile;
FIX32_1& operator/=(int b);
volatile FIX32_1& operator/=(float b) volatile;
FIX32_1& operator/=(float b);
volatile FIX32_1& operator/=(double b) volatile;
FIX32_1& operator/=(double b);

volatile FIX32_1& operator+=(FIX32_1 b) volatile;
FIX32_1& operator+=(FIX32_1 b);
volatile FIX32_1& operator-=(FIX32_1 b) volatile;
FIX32_1& operator-=(FIX32_1 b);

long value(){return val;}

// Other functions
friend FIX32_1 _fix321(long a);
friend long _bptn(FIX32_1 a);
friend float _float(FIX32_1 a);
friend double _double(FIX32_1 a);

friend FIX32_1 _abs(FIX32_1 a);
friend FIX32_1 _max(FIX32_1 a, FIX32_1 b);
friend FIX32_1 _min(FIX32_1 a, FIX32_1 b);
friend FIX32_1 _adds(FIX32_1 a, FIX32_1 b);
friend FIX32_1 _subs(FIX32_1 a, FIX32_1 b);
friend int _bcnt1(FIX32_1 a);
friend int _bseq(FIX32_1 a);
friend int _bseq0(FIX32_1 a);
friend int _bseq1(FIX32_1 a);
friend FIX16_1 _round(FIX32_1 a);

friend int _extr(FIX32_1 a, unsigned int b, unsigned int c);
friend unsigned int _extru(FIX32_1 a, unsigned int b, unsigned int c);
friend void _mul(long &m, long &m1, FIX32_1 &c, FIX32_1 a, FIX32_1 b);
friend void _mul(long &m, long &m1, FIX32_1 &c, FIX16_1 a, FIX16_1 b);
friend void _mul(long &m, long &m1, FIX32_1 &c, FIX16_1 a, FIX32_1 b);
friend void _mul(long &m, long &m1, FIX32_1 &c, FIX16_1 a, FIX16_1 b);

friend void _mac(long &m, long &m1, FIX32_1 &c, FIX16_1 a, FIX16_1 b);
friend void _mac(long &m, long &m1, FIX32_1 &c, FIX32_1 a, FIX32_1 b);
friend void _mac(long &m, long &m1, FIX32_1 &c, FIX32_1 a, FIX16_1 b);
friend void _mac(long &m, long &m1, FIX32_1 &c, FIX16_1 a, FIX32_1 b);

friend void _msu(long &m, long &m1, FIX32_1 &c, FIX32_1 a, FIX32_1 b);
friend void _msu(long &m, long &m1, FIX32_1 &c, FIX16_1 a, FIX16_1 b);
friend void _msu(long &m, long &m1, FIX32_1 &c, FIX32_1 a, FIX16_1 b);

```

【図43】

```

friend void _msu(long &nh, long&nl, FIX32_1 &c, FIX16_1 a, FIX32_1 b);
friend FIX32_1 _div(FIX32_1 a, FIX32_1 b);
friend FIX32_1 _div(FIX32_1 a, int b);
friend FIX32_1 _div(FIX32_1 a, float b);
friend FIX32_1 _div(FIX32_1 a, double b);

};

/////////////////////////////////////////////////////////////////
// Member of FIX16_2
//
// val : 実際の16bitの値
//
class FIX16_2{
    short val;
public:
    // constructor
    FIX16_2(){}
    FIX16_2(int a);
    FIX16_2(float a);
    FIX16_2(double a);
    FIX16_2(FIX16_2& a)
    {
        val = a.val;
    }
    FIX16_2(volatile FIX16_2& a)
    {
        val = a.val;
    }
    FIX16_2(const FIX16_2& a)
    {
        val = a.val;
    }

    // Operator
    volatile FIX16_2& operator=(FIX16_2 a) volatile
    {
        val = a.val;
        return *this;
    }
    FIX16_2& operator=(FIX16_2 a)
    {
        val = a.val;
        return *this;
    }
    friend FIX16_2 operator+(FIX16_2 a);
    friend FIX16_2 operator-(FIX16_2 a);

    friend FIX16_2 operator+(FIX16_2 a, FIX16_2 b);
    friend FIX16_2 operator-(FIX16_2 a, FIX16_2 b);

    friend FIX16_2 operator*(FIX16_2 a, FIX16_2 b);
    friend FIX16_2 operator*(int a, FIX16_2 b);
    friend FIX16_2 operator*(FIX16_2 a, int b);
    friend FIX16_2 operator*(float a, FIX16_2 b);
    friend FIX16_2 operator*(FIX16_2 a, float b);
    friend FIX16_2 operator*(double a, FIX16_2 b);
    friend FIX16_2 operator*(FIX16_2 a, double b);
}

```

【図44】

```

friend FIX16_2 operator/(FIX16_2 a, FIX16_2 b);
friend FIX16_2 operator/(FIX16_2 a, int b);
friend FIX16_2 operator/(FIX16_2 a, float b);
friend FIX16_2 operator/(FIX16_2 a, double b);

friend FIX16_2 operator<<(FIX16_2 a, int b);
friend FIX16_2 operator>>(FIX16_2 a, int b);

friend bool operator<(FIX16_2 a, FIX16_2 b);
friend bool operator>(FIX16_2 a, FIX16_2 b);
friend bool operator<=(FIX16_2 a, FIX16_2 b);
friend bool operator>=(FIX16_2 a, FIX16_2 b);
friend bool operator==(FIX16_2 a, FIX16_2 b);
friend bool operator!=(FIX16_2 a, FIX16_2 b);

volatile FIX16_2& operator<<=(int b) volatile;
    FIX16_2& operator<<=(int b);
volatile FIX16_2& operator>=(int b) volatile;
    FIX16_2& operator>=(int b);

volatile FIX16_2& operator*=(FIX16_2 b) volatile;
    FIX16_2& operator*=(FIX16_2 b);
volatile FIX16_2& operator*=(int b) volatile;
    FIX16_2& operator*=(int b);
volatile FIX16_2& operator*=(float b) volatile;
    FIX16_2& operator*=(float b);
volatile FIX16_2& operator*=(double b) volatile;
    FIX16_2& operator*=(double b);

volatile FIX16_2& operator/=(FIX16_2 b) volatile;
    FIX16_2& operator/=(FIX16_2 b);
volatile FIX16_2& operator/=(int b) volatile;
    FIX16_2& operator/=(int b);
volatile FIX16_2& operator/=(float b) volatile;
    FIX16_2& operator/=(float b);
volatile FIX16_2& operator/=(double b) volatile;
    FIX16_2& operator/=(double b);

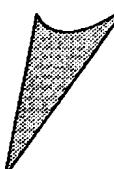
volatile FIX16_2& operator+=(FIX16_2 b) volatile;
    FIX16_2& operator+=(FIX16_2 b);
volatile FIX16_2& operator-=(FIX16_2 b) volatile;
    FIX16_2& operator-=(FIX16_2 b);

short value(){return val;}

// Other functions
friend FIX16_2 _fix162(short a);
friend short _bptn(FIX16_2 a);
friend FIX16_2 _fix162(FIX32_2 a);
friend float _float(FIX16_2 a);
friend double _double(FIX16_2 a);

friend FIX16_2 _abs(FIX16_2 a);
friend FIX16_2 _max(FIX16_2 a, FIX16_2 b);
friend FIX16_2 _min(FIX16_2 a, FIX16_2 b);
friend FIX16_2 _adds(FIX16_2 a, FIX16_2 b);
friend FIX16_2 _subs(FIX16_2 a, FIX16_2 b);

```



【図45】

```

friend int      _bcnt1(FIX16_2 a);
friend int      _bseq(FIX16_2 a);
friend int      _bseq0(FIX16_2 a);
friend int      _bseq1(FIX16_2 a);
friend FIX16_2  _round(FIX32_2 a);

friend int _extr(FIX16_2 a, unsigned int b, unsigned int c);
friend unsigned int _extru(FIX16_2 a, unsigned int b, unsigned int c);

friend void     _mul(long &mh, long&m, FIX16_2 &c, FIX16_2 a, FIX16_2 b);
friend void     _mul(long &mh, long&m, FIX32_2 &c, FIX16_2 a, FIX16_2 b);
friend void     _mul(long &mh, long&m, FIX32_2 &c, FIX16_2 a, FIX32_2 b);
friend void     _mul(long &mh, long&m, FIX32_2 &c, FIX32_2 a, FIX16_2 b);
friend void     _mac(long &mh, long &m, FIX16_2 &c, FIX16_2 a, FIX16_2 b);
friend void     _mac(long &mh, long &m, FIX32_2 &c, FIX16_2 a, FIX16_2 b);
friend void     _mac(long &mh, long &m, FIX32_2 &c, FIX32_2 a, FIX16_2 b);
friend void     _mac(long &mh, long &m, FIX32_2 &c, FIX32_2 a, FIX32_2 b);
friend void     _msu(long &mh, long&m, FIX16_2 &c, FIX16_2 a, FIX16_2 b);
friend void     _msu(long &mh, long&m, FIX32_2 &c, FIX16_2 a, FIX16_2 b);
friend void     _msu(long &mh, long&m, FIX32_2 &c, FIX32_2 a, FIX16_2 b);
friend void     _msu(long &mh, long&m, FIX32_2 &c, FIX32_2 a, FIX32_2 b);
friend FIX16_2  _div(FIX16_2 a, FIX16_2 b);
friend FIX16_2  _div(FIX16_2 a, int b);
friend FIX16_2  _div(FIX16_2 a, float b);
friend FIX16_2  _div(FIX16_2 a, double b);

};

/////////////////////////////////////////////////////////////////
// Member of FIX32_2
//
//          val : 実際の32bitの値
//
/////////////////////////////////////////////////////////////////
class FIX32_2{
    long val;
public:
    // constructor
    FIX32_2(){}
    FIX32_2(int a);
    FIX32_2(float a);
    FIX32_2(double a);
    FIX32_2(FIX16_2 a);
    FIX32_2(FIX32_2& a)
    {
        val = a.val;
    }
    FIX32_2(volatile FIX32_2& a)
    {
        val = a.val;
    }
    FIX32_2(const FIX32_2& a)
    {
        val = a.val;
    }

    // Operator
    volatile FIX32_2& operator=(FIX32_2 a) volatile
    {

```

【図46】

```

    val = a.val;
    return *this;
}
FIX32_2& operator=(FIX32_2 a)
{
    val = a.val;
    return *this;
}
friend FIX32_2 operator+(FIX32_2 a);
friend FIX32_2 operator-(FIX32_2 a);

friend FIX32_2 operator+(FIX32_2 a, FIX32_2 b);
friend FIX32_2 operator-(FIX32_2 a, FIX32_2 b);

friend FIX32_2 operator*(FIX32_2 a, FIX32_2 b);
friend FIX32_2 operator*(int a, FIX32_2 b);
friend FIX32_2 operator*(FIX32_2 a, int b);
friend FIX32_2 operator*(float a, FIX32_2 b);
friend FIX32_2 operator*(FIX32_2 a, float b);
friend FIX32_2 operator*(double a, FIX32_2 b);
friend FIX32_2 operator*(FIX32_2 a, double b);

friend FIX32_2 operator/(FIX32_2 a, FIX32_2 b);
friend FIX32_2 operator/(FIX32_2 a, int b);
friend FIX32_2 operator/(FIX32_2 a, float b);
friend FIX32_2 operator/(FIX32_2 a, double b);

friend FIX32_2 operator<<(FIX32_2 a, int b);
friend FIX32_2 operator>>(FIX32_2 a, int b);

friend bool operator<(FIX32_2 a, FIX32_2 b);
friend bool operator>(FIX32_2 a, FIX32_2 b);
friend bool operator<=(FIX32_2 a, FIX32_2 b);
friend bool operator>=(FIX32_2 a, FIX32_2 b);
friend bool operator==(FIX32_2 a, FIX32_2 b);
friend bool operator!=(FIX32_2 a, FIX32_2 b);

volatile FIX32_2& operator<=(int b) volatile;
    FIX32_2& operator<=(int b);
volatile FIX32_2& operator>=(int b) volatile;
    FIX32_2& operator>=(int b);

volatile FIX32_2& operator*=(FIX32_2 b) volatile;
    FIX32_2& operator*=(FIX32_2 b);
volatile FIX32_2& operator*=(int b) volatile;
    FIX32_2& operator*=(int b);
volatile FIX32_2& operator*=(float b) volatile;
    FIX32_2& operator*=(float b);
volatile FIX32_2& operator*=(double b) volatile;
    FIX32_2& operator*=(double b);

volatile FIX32_2& operator/=(FIX32_2 b) volatile;
    FIX32_2& operator/=(FIX32_2 b);
volatile FIX32_2& operator/=(int b) volatile;
    FIX32_2& operator/=(int b);
volatile FIX32_2& operator/=(float b) volatile;
    FIX32_2& operator/=(float b);
volatile FIX32_2& operator/=(double b) volatile;
    FIX32_2& operator/=(double b);

```

【図47】

```

volatile FIX32_2& operator+=(FIX32_2 b) volatile;
    FIX32_2& operator+=(FIX32_2 b);
volatile FIX32_2& operator-=(FIX32_2 b) volatile;
    FIX32_2& operator-=(FIX32_2 b);

long value(){return val;}

// Other functions
friend FIX32_2 _fix322(long a);
friend long _bptr(FIX32_2 a);
friend float _float(FIX32_2 a);
friend double _double(FIX32_2 a);

friend FIX32_2 _abs(FIX32_2 a);
friend FIX32_2 _max(FIX32_2 a, FIX32_2 b);
friend FIX32_2 _min(FIX32_2 a, FIX32_2 b);
friend FIX32_2 _adds(FIX32_2 a, FIX32_2 b);
friend FIX32_2 _subs(FIX32_2 a, FIX32_2 b);
friend int _bcnt1(FIX32_2 a);
friend int _bseq(FIX32_2 a);
friend int _bseq0(FIX32_2 a);
friend int _bseq1(FIX32_2 a);
friend FIX16_2 _round(FIX32_2 a);

friend int _extr(FIX32_2 a, unsigned int b, unsigned int c);
friend unsigned int _extru(FIX32_2 a, unsigned int b, unsigned int c);
friend void _mul(long &nh, long &ml, FIX32_2 &c, FIX32_2 a, FIX32_2 b);
friend void _mul(long &nh, long &ml, FIX32_2 &c, FIX16_2 a, FIX16_2 b);
friend void _mul(long &nh, long &ml, FIX32_2 &c, FIX16_2 a, FIX32_2 b);
friend void _mul(long &nh, long &ml, FIX32_2 &c, FIX32_2 a, FIX16_2 b);

friend void _mac(long &nh, long &ml, FIX32_2 &c, FIX16_2 a, FIX16_2 b);
friend void _mac(long &nh, long &ml, FIX32_2 &c, FIX32_2 a, FIX32_2 b);
friend void _mac(long &nh, long &ml, FIX32_2 &c, FIX32_2 a, FIX16_2 b);
friend void _mac(long &nh, long &ml, FIX32_2 &c, FIX16_2 a, FIX32_2 b);
friend void _msu(long &nh, long &ml, FIX32_2 &c, FIX32_2 a, FIX32_2 b);
friend void _msu(long &nh, long &ml, FIX32_2 &c, FIX16_2 a, FIX16_2 b);
friend void _msu(long &nh, long &ml, FIX32_2 &c, FIX32_2 a, FIX16_2 b);
friend void _msu(long &nh, long &ml, FIX32_2 &c, FIX16_2 a, FIX32_2 b);

friend FIX32_2 _div(FIX32_2 a, FIX32_2 b);
friend FIX32_2 _div(FIX32_2 a, int b);
friend FIX32_2 _div(FIX32_2 a, float b);
friend FIX32_2 _div(FIX32_2 a, double b);

};

#if defined(__AMMPCC__)
#pragma pack_struct_default
#endif // __AMMPCC__

// other functions
#if defined(__AMMPCC__)

#pragma _enable_asm_begin

static inline FIX16_1 _fix161(FIX32_1 a)
{
    FIX16_1 result;

```

【図48】

```
asm(vr0 = a){  
    asr vr1, vr0, 16;  
}(result = vr1);  
  
    return result;  
}  
  
static inline FIX16_2 _fix162(FIX32_2 a)  
{  
    FIX16_2 result;  
  
    asm(vr0 = a){  
        asr vr1, vr0, 16;  
    }(result = vr1);  
  
    return result;  
}  
  
static inline FIX16_1 _fix161(short a)  
{  
    FIX16_1 result;  
  
    asm(vr0 = a){  
        mov vr1, vr0;  
    }(result = vr1);  
  
    return result;  
}  
  
static inline FIX16_2 _fix162(short a)  
{  
    FIX16_2 result;  
  
    asm(vr0 = a){  
        mov vr1, vr0;  
    }(result = vr1);  
  
    return result;  
}  
  
static inline FIX32_1 _fix321(long a)  
{  
    FIX32_1 result;  
  
    asm(vr0 = a){  
        mov vr1, vr0;  
    }(result = vr1);  
  
    return result;  
}  
  
static inline FIX32_2 _fix322(long a)  
{  
    FIX32_2 result;  
  
    asm(vr0 = a){  
        mov vr1, vr0;  
    }(result = vr1);  
}
```

【図49】

```
        return result;
    }

static inline short _bptn(FIX16_1 a)
{
    short result;

    asm(vr0 = a){
        mov vr1, vr0;
    }(result = vr1);

    return result;
}

static inline short _bptn(FIX16_2 a)
{
    short result;

    asm(vr0 = a){
        mov vr1, vr0;
    }(result = vr1);

    return result;
}

static inline long _bptn(FIX32_1 a)
{
    long result;

    asm(vr0 = a){
        mov vr1, vr0;
    }(result = vr1);

    return result;
}

static inline long _bptn(FIX32_2 a)
{
    long result;

    asm(vr0 = a){
        mov vr1, vr0;
    }(result = vr1);

    return result;
}

static inline FIX16_1 _abs(FIX16_1 a)
{
    FIX16_1 result;

    asm(vr0 = a){
        absvh vr1, vr0;
    }(result = vr1);

    return result;
}
```

【図50】

```
static inline FIX32_1 _abs(FIX32_1 a)
{
    FIX32_1 result;
    asm(vr0 = a){
        absvw vr1, vr0;
    }(result = vr1);
    return result;
}

static inline FIX16_2 _abs(FIX16_2 a)
{
    FIX16_2 result;
    asm(vr0 = a){
        absvh vr1, vr0;
    }(result = vr1);
    return result;
}

static inline FIX32_2 _abs(FIX32_2 a)
{
    FIX32_2 result;
    asm(vr0 = a){
        absvw vr1, vr0;
    }(result = vr1);
    return result;
}

static inline FIX16_1 _max(FIX16_1 a, FIX16_1 b)
{
    FIX16_1 result;
    asm(vr0 = a, vr1 = b){
        max vr2, vr0, vr1;
    }(result = vr2);
    return result;
}

static inline FIX32_1 _max(FIX32_1 a, FIX32_1 b)
{
    FIX32_1 result;
    asm(vr0 = a, vr1 = b){
        max vr2, vr0, vr1;
    }(result = vr2);
    return result;
}

static inline FIX16_2 _max(FIX16_2 a, FIX16_2 b)
{
    FIX16_2 result;
```

【図51】

```
asm(vr0 = a, vr1 = b){  
    max vr2, vr0, vr1;  
}(result = vr2);  
  
    return result;  
}  
  
static inline FIX32_2 _max(FIX32_2 a, FIX32_2 b)  
{  
    FIX32_2 result;  
  
    asm(vr0 = a, vr1 = b){  
        max vr2, vr0, vr1;  
    }(result = vr2);  
  
    return result;  
}  
  
static inline FIX16_1 _min(FIX16_1 a, FIX16_1 b)  
{  
    FIX16_1 result;  
  
    asm(vr0 = a, vr1 = b){  
        min vr2, vr0, vr1;  
    }(result = vr2);  
  
    return result;  
}  
  
static inline FIX32_1 _min(FIX32_1 a, FIX32_1 b)  
{  
    FIX32_1 result;  
  
    asm(vr0 = a, vr1 = b){  
        min vr2, vr0, vr1;  
    }(result = vr2);  
  
    return result;  
}  
  
static inline FIX16_2 _min(FIX16_2 a, FIX16_2 b)  
{  
    FIX16_2 result;  
  
    asm(vr0 = a, vr1 = b){  
        min vr2, vr0, vr1;  
    }(result = vr2);  
  
    return result;  
}  
  
static inline FIX32_2 _min(FIX32_2 a, FIX32_2 b)  
{  
    FIX32_2 result;  
  
    asm(vr0 = a, vr1 = b){  
        min vr2, vr0, vr1;
```

【図52】

```
}(result = vr2);
return result;
}

static inline FIX16_1 _adds(FIX16_1 a, FIX16_1 b)
{
    FIX16_1 result;
    asm(vr0 = a, vr1 = b){
        adds    vr2, vr0, vr1;
    }(result = vr2);
    return result;
}

static inline FIX32_1 _adds(FIX32_1 a, FIX32_1 b)
{
    FIX32_1 result;
    asm(vr0 = a, vr1 = b){
        adds    vr2, vr0, vr1;
    }(result = vr2);
    return result;
}

static inline FIX16_2 _adds(FIX16_2 a, FIX16_2 b)
{
    FIX16_2 result;
    asm(vr0 = a, vr1 = b){
        adds    vr2, vr0, vr1;
    }(result = vr2);
    return result;
}

static inline FIX32_2 _adds(FIX32_2 a, FIX32_2 b)
{
    FIX32_2 result;
    asm(vr0 = a, vr1 = b){
        adds    vr2, vr0, vr1;
    }(result = vr2);
    return result;
}

static inline FIX16_1 _subs(FIX16_1 a, FIX16_1 b)
{
    FIX16_1 result;
    asm(vr0 = a, vr1 = b){
        subs    vr2, vr0, vr1;
    }(result = vr2);
    return result;
}
```

【図53】

```
static inline FIX32_1 _subs(FIX32_1 a, FIX32_1 b)
{
    FIX32_1 result;

    asm(vr0 = a, vr1 = b){
        subs    vr2, vr0, vr1;
    }(result = vr2);

    return result;
}

static inline FIX16_2 _subs(FIX16_2 a, FIX16_2 b)
{
    FIX16_2 result;

    asm(vr0 = a, vr1 = b){
        subs    vr2, vr0, vr1;
    }(result = vr2);

    return result;
}

static inline FIX32_2 _subs(FIX32_2 a, FIX32_2 b)
{
    FIX32_2 result;

    asm(vr0 = a, vr1 = b){
        subs    vr2, vr0, vr1;
    }(result = vr2);

    return result;
}

static inline int _bcnt1(FIX16_1 a)
{
    int result;

    asm(vr0 = a){
        bcnt1   vr1, vr0;
    }(result = vr1);

    return result;
}

static inline int _bcnt1(FIX16_2 a)
{
    int result;

    asm(vr0 = a){
        bcnt1   vr1, vr0;
    }(result = vr1);

    return result;
}

static inline int _bcnt1(FIX32_1 a)
{
    int result;
```

【図54】

```
asm(vr0 = a){  
    bcnt1    vr1, vr0;  
}(result = vr1);  
  
    return result;  
}  
  
static inline int _bcnt1(FIX32_2 a)  
{  
    int result;  
  
    asm(vr0 = a){  
        bcnt1    vr1, vr0;  
    }(result = vr1);  
  
    return result;  
}  
  
static inline int _bseq(FIX16_1 a)  
{  
    int result;  
  
    asm(vr0 = a){  
        bseq     vr1, vr0;  
    }(result = vr1);  
  
    return result;  
}  
  
static inline int _bseq0(FIX16_1 a)  
{  
    int result;  
  
    asm(vr0 = a){  
        bseq0    vr1, vr0;  
    }(result = vr1);  
  
    return result;  
}  
  
static inline int _bseq1(FIX16_1 a)  
{  
    int result;  
  
    asm(vr0 = a){  
        bseq1    vr1, vr0;  
    }(result = vr1);  
  
    return result;  
}  
  
static inline int _bseq(FIX32_1 a)  
{  
    int result;  
  
    asm(vr0 = a){  
        bseq     vr1, vr0;  
    }(result = vr1);  
}
```

【図55】

```
    return result;
}

static inline int _bseq0(FIX32_1 a)
{
    int result;

    asm(vr0 = a){
        bseq0    vr1, vr0;
    }(result = vr1);

    return result;
}

static inline int _bseq1(FIX32_1 a)
{
    int result;

    asm(vr0 = a){
        bseq1    vr1, vr0;
    }(result = vr1);

    return result;
}

static inline int _bseq(FIX16_2 a)
{
    int result;

    asm(vr0 = a){
        bseq    vr1, vr0;
    }(result = vr1);

    return result;
}

static inline int _bseq0(FIX16_2 a)
{
    int result;

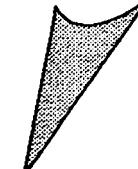
    asm(vr0 = a){
        bseq0    vr1, vr0;
    }(result = vr1);

    return result;
}

static inline int _bseq1(FIX16_2 a)
{
    int result;

    asm(vr0 = a){
        bseq1    vr1, vr0;
    }(result = vr1);

    return result;
}
```



【図56】

```

static inline int _bseq(FIX32_2 a)
{
    int result;
    asm(vr0 = a){
        bseq    vr1, vr0;
    }(result = vr1);
    return result;
}

static inline int _bseq0(FIX32_2 a)
{
    int result;
    asm(vr0 = a){
        bseq0   vr1, vr0;
    }(result = vr1);
    return result;
}

static inline int _bseq1(FIX32_2 a)
{
    int result;
    asm(vr0 = a){
        bseq1   vr1, vr0;
    }(result = vr1);
    return result;
}

static inline FIX16_1 _round(FIX32_1 a)
{
    FIX16_1 result;
    asm(vr0 = a){
        rndvh  vr1, vr0;
    }(result = vr1);
    return result;
}

static inline FIX16_2 _round(FIX32_2 a)
{
    FIX16_2 result;
    asm(vr0 = a){
        rndvh  vr1, vr0;
    }(result = vr1);
    return result;
}

static inline void _mulr(FIX16_1 &c, FIX16_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b) {
        fmulhhr m0,vr2,vr0,vr1;
}

```

## 【図57】

```

    } (c = vr2);

static inline void _mulr(long &mh, long&m1, FIX16_1 &c, FIX16_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b) {
        fmulhh m0,vr2,vr0,vr1;
    } (mh = mh0, m1 = m10, c = vr2);
}

static inline void _mul(long &mh, long&m1, FIX16_1 &c, FIX16_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b) {
        fmulhh m0,vr2,vr0,vr1;
    } (mh = mh0, m1 = m10, c = vr2);
}

static inline void _mul(long &mh, long&m1, FIX16_2 &c, FIX16_2 a, FIX16_2 b)
{
    asm(vr0 = a, vr1 = b) {
        fmulhh m0,vr2,vr0,vr1;
    } (mh = mh0, m1 = m10, c = vr2);
}

static inline void _mul(long &mh, long&m1, FIX32_1 &c, FIX16_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b) {
        fmulhw m0,vr2,vr0,vr1;
    } (mh = mh0, m1 = m10, c = vr2);
}

static inline void _mul(long &mh, long&m1, FIX32_2 &c, FIX16_2 a, FIX16_2 b)
{
    asm(vr0 = a, vr1 = b) {
        fmulhw m0,vr2,vr0,vr1;
    } (mh = mh0, m1 = m10, c = vr2);
}

static inline void _mul(long &mh, long&m1, FIX32_1 &c, FIX32_1 a, FIX32_1 b)
{
    asm(vr0 = a, vr1 = b) {
        fmulww m0,vr2,vr0,vr1;
    } (mh = mh0, m1 = m10, c = vr2);
}

static inline void _mul(long &mh, long&m1, FIX32_2 &c, FIX32_2 a, FIX32_2 b)
{
    asm(vr0 = a, vr1 = b) {
        fmulww m0,vr2,vr0,vr1;
    } (mh = mh0, m1 = m10, c = vr2);
}

static inline void _mul(long &mh, long&m1, FIX32_1 &c, FIX16_1 a, FIX32_1 b)
{
    asm(vr0 = a, vr1 = b) {
        fmulhww m0,vr2,vr1,vr0;
    } (mh = mh0, m1 = m10, c = vr2);
}

```

【図58】

```

static inline void _mul(long &mh, long&m1, FIX32_2 &c, FIX16_2 a, FIX32_2 b)
{
    asm(vr0 = a, vr1 = b) {
        fmulhww m0,vr2,vr1,vr0;
    } (mh = mh0, m1 = m10, c = vr2);
}

static inline void _mul(long &mh, long&m1, FIX32_1 &c, FIX32_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b) {
        fmulhww m0,vr2,vr0,vr1;
    } (mh = mh0, m1 = m10, c = vr2);
}

static inline void _mul(long &mh, long&m1, FIX32_2 &c, FIX32_2 a, FIX16_2 b)
{
    asm(vr0 = a, vr1 = b) {
        fmulhww m0,vr2,vr0,vr1;
    } (mh = mh0, m1 = m10, c = vr2);
}

static inline void _mac(long &mh, long &m1, FIX16_1 &c, FIX16_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, m10 = m1) {
        fmachh m0,vr2,vr0,vr1,m0;
    } (mh = mh0, m1 = m10, c = vr2);
}

static inline void _mac(long &mh, long &m1, FIX16_2 &c, FIX16_2 a, FIX16_2 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, m10 = m1) {
        fmachh m0,vr2,vr0,vr1,m0;
    } (mh = mh0, m1 = m10, c = vr2);
}

static inline void _mac(long &mh, long &m1, FIX32_1 &c, FIX16_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, m10 = m1) {
        fmachw m0,vr2,vr0,vr1,m0;
    } (mh = mh0, m1 = m10, c = vr2);
}

static inline void _mac(long &mh, long &m1, FIX32_2 &c, FIX16_2 a, FIX16_2 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, m10 = m1) {
        fmachw m0,vr2,vr0,vr1,m0;
    } (mh = mh0, m1 = m10, c = vr2);
}

static inline void _mac(long &mh, long &m1, FIX32_1 &c, FIX32_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, m10 = m1) {
        fmachww m0,vr2,vr0,vr1,m0;
    } (mh = mh0, m1 = m10, c = vr2);
}

static inline void _mac(long &mh, long &m1, FIX32_2 &c, FIX32_2 a, FIX16_2 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, m10 = m1) {

```

【図59】

```

        fmacchww m0,vr2,vr0,vr1,m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _mac(long &mh, long &ml, FIX32_1 &c, FIX16_1 a, FIX32_1 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmacchww m0,vr2,vr1,vr0,m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _mac(long &mh, long &ml, FIX32_2 &c, FIX16_2 a, FIX32_2 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmacchww m0,vr2,vr1,vr0,m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _mac(long &mh, long &ml, FIX32_1 &c, FIX32_1 a, FIX32_1 b)
{
    asm(vr0 = b, vr1 = a, mh0 = mh, ml0 = ml) {
        fmacchww m0,vr2,vr0,vr1,m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _mac(long &mh, long &ml, FIX32_2 &c, FIX32_2 a, FIX32_2 b)
{
    asm(vr0 = b, vr1 = a, mh0 = mh, ml0 = ml) {
        fmacchww m0,vr2,vr0,vr1,m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _macr(long &mh, long &ml, FIX16_1 &c, FIX16_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmacchhr m0,vr2,vr0,vr1,m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _msu(long &mh, long &ml, FIX16_1 &c, FIX16_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmsuhh m0,vr2,vr0,vr1,m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _msu(long &mh, long &ml, FIX16_2 &c, FIX16_2 a, FIX16_2 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmsuhh m0,vr2,vr0,vr1,m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

static inline void _msu(long &mh, long &ml, FIX32_1 &c, FIX16_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, ml0 = ml) {
        fmsuhw m0,vr2,vr0,vr1,m0;
    } (mh = mh0, ml = ml0, c = vr2);
}

```

【図60】

```

static inline void _msu(long &mh, long &m1, FIX32_2 &c, FIX16_2 a, FIX16_2 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, m10 = m1) {
        fmsuhw m0,vr2,vr0,vr1,m0;
    } (mh = mh0, m1 = m10, c = vr2);
}

static inline void _msu(long &mh, long &m1, FIX32_1 &c, FIX32_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, m10 = m1) {
        fmsuhww m0,vr2,vr0,vr1,m0;
    } (mh = mh0, m1 = m10, c = vr2);
}

static inline void _msu(long &mh, long &m1, FIX32_2 &c, FIX32_2 a, FIX16_2 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, m10 = m1) {
        fmsuhww m0,vr2,vr0,vr1,m0;
    } (mh = mh0, m1 = m10, c = vr2);
}

static inline void _msu(long &mh, long &m1, FIX32_1 &c, FIX16_1 a, FIX32_1 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, m10 = m1) {
        fmsuhww m0,vr2,vr1,vr0,m0;
    } (mh = mh0, m1 = m10, c = vr2);
}

static inline void _msu(long &mh, long &m1, FIX32_2 &c, FIX16_2 a, FIX32_2 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, m10 = m1) {
        fmsuhww m0,vr2,vr1,vr0,m0;
    } (mh = mh0, m1 = m10, c = vr2);
}

static inline void _msu(long &mh, long &m1, FIX32_1 &c, FIX32_1 a, FIX32_1 b)
{
    asm(vr0 = b, vr1 = a, mh0 = mh, m10 = m1) {
        fmsuhww m0,vr2,vr0,vr1,m0;
    } (mh = mh0, m1 = m10, c = vr2);
}

static inline void _msu(long &mh, long &m1, FIX32_2 &c, FIX32_2 a, FIX32_2 b)
{
    asm(vr0 = b, vr1 = a, mh0 = mh, m10 = m1) {
        fmsuhww m0,vr2,vr0,vr1,m0;
    } (mh = mh0, m1 = m10, c = vr2);
}

static inline void _msur(long &mh, long &m1, FIX16_1 &c, FIX16_1 a, FIX16_1 b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, m10 = m1) {
        fmsuhhr m0,vr2,vr0,vr1,m0;
    } (mh = mh0, m1 = m10, c = vr2);
}

inline FIX16_1 operator/(FIX16_1 a, FIX16_1 b)
{
}

```

【図61】

```

FIX16_1 result;
asm(vr0 = a, vr1 = b) {
    extw m0,vr3,vr0;
    aslp m0,vr4,mh0,vr3,15;
    div mh1,vr5,mh0,vr4,vr1;
    sath vr2,vr5;
} (result = vr2);

return result;
}

inline FIX16_2 operator/(FIX16_2 a, FIX16_2 b)
{
    FIX16_2 result;
    asm(vr0 = a, vr1 = b) {
        extw m0,vr3,vr0;
        aslp m0,vr4,mh0,vr3,14;
        div mh1,vr5,mh0,vr4,vr1;
        sath vr2,vr5;
    } (result = vr2);

    return result;
}

inline FIX32_1 operator/(FIX32_1 a, FIX32_1 b)
{
    FIX32_1 result;
    asm(vr0 = a, vr1 = b) {
        mskgen vr3,31,31; //vr3 = 0x80000000
        cmpeq C0:C1,vr1,vr3;
        [C0] negvw vr2,vr0; //vr1==(-1)の場合、符号反転
        cmpgtn C2:C3,vr0,0,C1; //vr1!=(-1)の場合
        [C2] negvw vr4,vr0; //vr0を負に符号反転(vr0')
        mov vr4,vr0;
        cmpgtn C2:C3,vr1,0,C1;
        [C2] negvw vr5,vr1; //vr1を負に符号反転(vr1')
        [C3] mov vr5,vr1;
        cmplen C2:C3,vr4,vr5,C1;
        [C2] lsr vr6,vr0,31; //vr0' <= 2vr1' (ovf 判定)の場合オーバーフロ
        - [C2] lsr vr7,vr1,31;
        [C2] xor vr8,vr6,vr7;
        cmpeqn C4:C5,vr8,0,C2;
        [C4] mskgen vr2,30,0; //vr2 = 0x7fffffff
        [C5] mskgen vr2,31,31; //vr2 = 0x80000000
        [C3] extw m0,vr9,vr0; //被除数<除数の場合
        [C3] aslp m0,vr10,mh0,vr9,31;
        [C3] div mh1,vr2,mh0,vr10,vr1;
    } (result = vr2);

    return result;
}

```

【図62】

```

inline FIX32_2 operator/(FIX32_2 a, FIX32_2 b)
{
    FIX32_2 result;

    asm(vr0 = a, vr1 = b) {
        mskgen vr3,31,31; //vr3 = 0x80000000
        mskgen vr4,31,30; //vr4 = 0xc0000000
        cmplt C0:C1,vr0,0;
        [C0] negvw vr5,vr0; //vr0を負に符号反転(vr0')
        [C1] mov vr5,vr0;
        cmplt C0:C1,vr1,0;
        [C0] negvw vr6,vr1; //vr1を負に符号反転(vr1')
        [C1] mov vr6,vr1;
        cmpeq C0:C1,vr0,vr3;
        cmplta C0:C1,vr6,vr4,C0;//vr0=-2かつvr1'<0xc0000000か?
        [C1] asl vw vr6,vr6,1; //!(vr0=-2かつvr1'<0xc0000000)のとき
        cmplea C2:C3,vr5,vr6,C1;
        [C2] lsr vr7,vr0,31; //!(vr0=-2かつvr1'<0xc0000000)かつ
        [C2] lsr vr8,vr1,31; // vr0' <= 2vr1'(ovf判定)の場合
        [C2] xor vr9,vr7,vr8; //オーバーフロー
        cmpeqn C4:C5,vr9,0,C2;
        [C4] mskgen vr2,30,0; //vr2 = 0x7fffffff
        [C5] mskgen vr2,31,31; //vr2 = 0x80000000
        extw m0,vr3,vr0; //それ以外
        [C3] aslp m0,vr10,vr0;
        [C3] div mh1,vr2,mh0,vr3,vr1;
    } (result = vr2);

    return result;
}

inline FIX16_1 operator/(FIX16_1 a, int b)
{
    FIX16_1 result;

    asm(vr0 = a, vr1 = b) {
        extw m0,vr3,vr0;
        div mh1,vr4,mh0,vr3,vr1;
        sath vr2,vr4;
    } (result = vr2);

    return result;
}

inline FIX16_2 operator/(FIX16_2 a, int b)
{
    FIX16_2 result;

    asm(vr0 = a, vr1 = b) {
        extw m0,vr3,vr0;
        div mh1,vr4,mh0,vr3,vr1;
        sath vr2,vr4;
    } (result = vr2);

    return result;
}

```

【図63】

```

inline FIX32_1 operator/(FIX32_1 a, int b)
{
    FIX32_1 result;

    asm(vr0 = a, vr1 = b) {
        cmpeq C0:C1,vr1,1;
        [C0] mov vr2,vr0; //除数=1の時、Rc = Ra
        [C1] mskgen vr3,31,31; //0x80000000
        cmpeqn C2:C3,vr0,vr3,C1;
        cmpeqa C4:C5,vr1,-1,C2;
        [C4] mskgen vr2,30,0; //被除数=-1かつ除数=-1の時、Rc =
0x7fffffff
        [C5] extw m0,vr4,vr0; //それ以外
        [C5] div mh1,vr2,mh0,vr4,vr1;
    } (result = vr2);

    return result;
}

inline FIX32_2 operator/(FIX32_2 a, int b)
{
    FIX32_2 result;

    asm(vr0 = a, vr1 = b) {
        mskgen vr3,31,31; //Rd = 0x80000000
        mskgen vr4,31,30; //Re = 0xc0000000
        cmpgt C0:C1,vr0,0;
        [C0] negvw vr5,vr0; //Raを負に符号反転(Ra')
        [C1] mov vr5,vr0;
        asl vw vr6,vr1,30; //int-->FIX32_2
        cmpgt C0:C1,vr6,0;
        [C0] negvw vr7,vr6; //Rbを負に符号反転(Rb')
        [C1] mov vr7,vr6;
        cmpeq C0:C1,vr0,vr3;
        [C1] cmplta C0:C1,vr7,vr4,C0; //Ra=-2かつRb'<0xc0000000か?
        asl vw vr7,vr7,1; //!(Ra=-2かつRb'<0xc0000000)のとき
        cmplea C2:C3,vr5,vr7,C1;
        [C2] lsr vr8,vr0,31; //!(Ra=-2かつRb'<0xc0000000)かつ
        [C2] lsr vr9,vr1,31; // Ra' <= 2Rb' (ovf判定)の場合
        [C2] xor vr10,vr8,vr9; //オーバーフロー
        cmpeqn C4:C5,vr10,0,C2;
        [C4] mskgen vr2,30,0; //Rc = 0x7fffffff
        [C5] mskgen vr2,31,31; //Rc = 0x80000000
        [C3] extw m0,vr11,vr0; //それ以外
        [C3] div mh1,vr2,mh0,vr11,vr1;
    } (result = vr2);

    return result;
}

inline FIX16_1 operator/(FIX16_1 a, float b)
{
    FIX16_1 c = b;
    return a/c;
}

```

【図64】

```

inline FIX16_1 operator/(FIX16_1 a, double b)
{
    FIX16_1 c = b;
    return a/c;
}

inline FIX32_1 operator/(FIX32_1 a, float b)
{
    FIX32_1 c = b;
    return a/c;
}

inline FIX32_1 operator/(FIX32_1 a, double b)
{
    FIX32_1 c = b;
    return a/c;
}

inline FIX16_2 operator/(FIX16_2 a, float b)
{
    FIX16_2 c = b;
    return a/c;
}

inline FIX16_2 operator/(FIX16_2 a, double b)
{
    FIX16_2 c = b;
    return a/c;
}

inline FIX32_2 operator/(FIX32_2 a, float b)
{
    FIX32_2 c = b;
    return a/c;
}

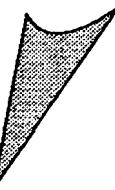
inline FIX32_2 operator/(FIX32_2 a, double b)
{
    FIX32_2 c = b;
    return a/c;
}

inline FIX16_1& FIX16_1::operator/=(FIX16_1 b)
{
    *this = *this / b;
    return *this;
}

inline volatile FIX16_1& FIX16_1::operator/=(FIX16_1 b) volatile
{
    *this = *this / b;
    return *this;
}

inline FIX16_1& FIX16_1::operator/=(int b)
{
    *this = *this / b;
    return *this;
}

```



【図65】

```
inline volatile FIX16_1& FIX16_1::operator/=(int b) volatile
{
    *this = *this / b;
    return *this;
}

inline FIX16_1& FIX16_1::operator/=(float b)
{
    *this = *this / b;
    return *this;
}

inline volatile FIX16_1& FIX16_1::operator/=(float b) volatile
{
    *this = *this / b;
    return *this;
}

inline FIX16_1& FIX16_1::operator/=(double b)
{
    *this = *this / b;
    return *this;
}

inline volatile FIX16_1& FIX16_1::operator/=(double b) volatile
{
    *this = *this / b;
    return *this;
}

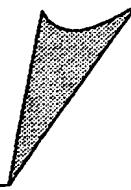
inline FIX32_1& FIX32_1::operator/=(FIX32_1 b)
{
    *this = *this / b;
    return *this;
}

inline volatile FIX32_1& FIX32_1::operator/=(FIX32_1 b) volatile
{
    *this = *this / b;
    return *this;
}

inline FIX32_1& FIX32_1::operator/=(int b)
{
    *this = *this / b;
    return *this;
}

inline volatile FIX32_1& FIX32_1::operator/=(int b) volatile
{
    *this = *this / b;
    return *this;
}

inline FIX32_1& FIX32_1::operator/=(float b)
{
    *this = *this / b;
    return *this;
}
```



【図66】

```
}

inline volatile FIX32_1& FIX32_1::operator/=(float b) volatile
{
    *this = *this / b;
    return *this;
}

inline FIX32_1& FIX32_1::operator/=(double b)
{
    *this = *this / b;
    return *this;
}

inline volatile FIX32_1& FIX32_1::operator/=(double b) volatile
{
    *this = *this / b;
    return *this;
}

inline FIX16_2& FIX16_2::operator/=(FIX16_2 b)
{
    *this = *this / b;
    return *this;
}

inline volatile FIX16_2& FIX16_2::operator/=(FIX16_2 b) volatile
{
    *this = *this / b;
    return *this;
}

inline FIX16_2& FIX16_2::operator/=(int b)
{
    *this = *this / b;
    return *this;
}

inline volatile FIX16_2& FIX16_2::operator/=(int b) volatile
{
    *this = *this / b;
    return *this;
}

inline FIX16_2& FIX16_2::operator/=(float b)
{
    *this = *this / b;
    return *this;
}

inline volatile FIX16_2& FIX16_2::operator/=(float b) volatile
{
    *this = *this / b;
    return *this;
}

inline FIX16_2& FIX16_2::operator/=(double b)
{
    *this = *this / b;
```

【図67】

```
    return *this;
}

inline volatile FIX16_2& FIX16_2::operator/=(double b) volatile
{
    *this = *this / b;
    return *this;
}

inline FIX32_2& FIX32_2::operator/=(FIX32_2 b)
{
    *this = *this / b;
    return *this;
}

inline volatile FIX32_2& FIX32_2::operator/=(FIX32_2 b) volatile
{
    *this = *this / b;
    return *this;
}

inline FIX32_2& FIX32_2::operator/=(int b)
{
    *this = *this / b;
    return *this;
}

inline volatile FIX32_2& FIX32_2::operator/=(int b) volatile
{
    *this = *this / b;
    return *this;
}

inline FIX32_2& FIX32_2::operator/=(float b)
{
    *this = *this / b;
    return *this;
}

inline volatile FIX32_2& FIX32_2::operator/=(float b) volatile
{
    *this = *this / b;
    return *this;
}

inline FIX32_2& FIX32_2::operator/=(double b)
{
    *this = *this / b;
    return *this;
}

inline volatile FIX32_2& FIX32_2::operator/=(double b) volatile
{
    *this = *this / b;
    return *this;
}

#pragma _enable_asm_end
```

【図68】

```
#endif //__AMMPCC__
#endif //__FIXAMMP__
```

【図69】

```

*****
*
* (C)Copyright 2002 Matsushita Electric Industrial Co., Ltd.
*   funcammp.h
*   Version:
*   Release:
*   Date:      2002/6/19
*
*****
```

```

/* 多重定義回避 */
#ifndef __FUNCAMMP__
#define __FUNCAMMP__
```

```

#include <stdlib.h>
```

```

#ifndef __AMMPCC__
```

```

long _abs(long);
long _max(long, long);
long _min(long, long);
long _adds(long, long);
long _subs(long, long);
int _bcnt1(long);
int _bseq0(long);
int _bseq1(long);
int _bseq(long);
int _log2(size_t size);
long _extr(long, unsigned int, unsigned int);
unsigned long _extru(long, unsigned int, unsigned int);
void _clrm(long&, long&);
void _mul(long&, long&, long&, long, long);
void _mac(long&, long&, long&, long, long);
void _msu(long&, long&, long&, long, long);
void *_modulo_add(void *, int, int, size_t, void *);
void *_brev_add(void *, int, int, int, size_t, void *);
```

```

#else /* defined __AMMPCC__ */
```

```

#pragma _enable_asm_begin
#pragma _enable_inline_begin
```

```

long _extr(long, unsigned int, unsigned int);
unsigned long _extru(long, unsigned int, unsigned int);
int _log2(size_t size);
```

```

static inline long
_abs(long data)
{
    long result;

    asm(vr0 = data){
        abs vr1, vr0;
    }(result = vr1);

    return result;
}
```

【図70】

```
static inline long
max(long data1, long data2)
{
    long result;
    asm(vr0 = data1, vr1 = data2){
        max vr2, vr1, vr0;
    }(result = vr2);
    return result;
}

static inline long
min(long data1, long data2)
{
    long result;
    asm(vr0 = data1, vr1 = data2){
        min vr2, vr1, vr0;
    }(result = vr2);
    return result;
}

static inline long
adds(long data1, long data2)
{
    long result;
    asm(vr0 = data1, vr1 = data2){
        adds vr2, vr0, vr1;
    }(result = vr2);
    return result;
}

static inline long
subs(long data1, long data2)
{
    long result;
    asm(vr0 = data1, vr1 = data2){
        subs vr2, vr0, vr1;
    }(result = vr2);
    return result;
}

static inline int
bcnt1(long data)
{
    long result;
    asm(vr0 = data){
        bcnt1 vr1, vr0;
    }(result = vr1);
    return result;
}
```

【図71】

```

static inline int
bseq0(long data)
{
    long result;
    asm(vr0 = data){
        bseq0 vr1, vr0;
    }(result = vr1);
    return result;
}

static inline int
bseq1(long data)
{
    long result;
    asm(vr0 = data){
        bseq1 vr1, vr0;
    }(result = vr1);
    return result;
}

static inline int
bseq(long data)
{
    long result;
    asm(vr0 = data){
        bseq vr1, vr0;
    }(result = vr1);
    return result;
}

static inline void
clrm(long &mh, long &m1)
{
    asm() {
        mul m0, vr1, vr0, 0;
    } (mh = mh0, m1 = m10);
}

static inline void
mui(long &mh, long&m1, long &c, long a, long b)
{
    asm(vr0 = a, vr1 = b) {
        mul m0, vr2, vr0, vr1;
    } (mh = mh0, m1 = m10, c = vr2);
}

static inline void
mac(long &mh, long &m1, long &c, long a, long b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, m10 = m1) {
        mac m0, vr2, vr0, vr1, m0;
    } (mh = mh0, m1 = m10, c = vr2);
}

```

【図72】

```

}

static inline void
_msu(long &mh, long &m1, long &c, long a, long b)
{
    asm(vr0 = a, vr1 = b, mh0 = mh, m10 = m1) {
        msu m0,vr2,vr0,vr1,m0;
    } (mh = mh0, m1 = m10, c = vr2);
}

static inline void *
modulo_add(void *addr, int imm, int mask, size_t size, void *base)
{
    void *p;
    int tmp1, tmp2, tmp3;

    tmp1 = _log2(size);
    tmp2 = mask + tmp1 - 1;
    tmp3 = imm << tmp1;

    asm(vr0 = addr, vr2 = base, vr3 = tmp2, vr4 = tmp3) {
        mov    CFR0,vr3;
        add    vr6,vr0,vr4;
        addmsk vr7,vr2,vr6;
    } (p = vr7);

    return p;
}

static inline void *
_brev_add(void *addr, int cnt, int imm, int mask, size_t size, void *base)
{
    void *p;
    int tmp1, tmp2, tmp3, tmp4;

    tmp1 = _log2(size);
    tmp2 = mask + tmp1 - 1;
    tmp3 = 16 - mask - tmp1;
    tmp4 = imm << tmp3;

    asm(vr0 = addr, vr1 = cnt, vr2 = base, vr3 = tmp2, vr4 = tmp3, vr5 = tmp4) {
        mov    CFR0,vr3;
        lsl    vr6,vr1,vr4;
        add    vr7,vr6,vr5;
        mskbrvh vr8,vr2,vr7;
    } (p = vr8);

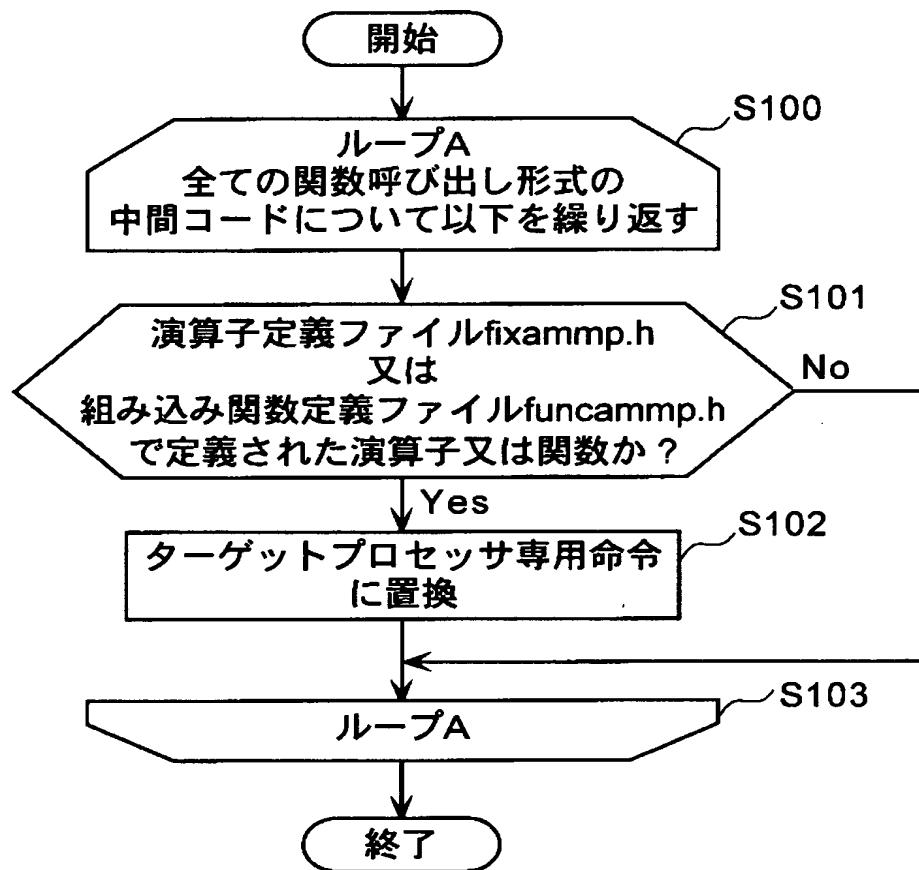
    return p;
}

#pragma _enable_inline_end
#pragma _enable_asm_end

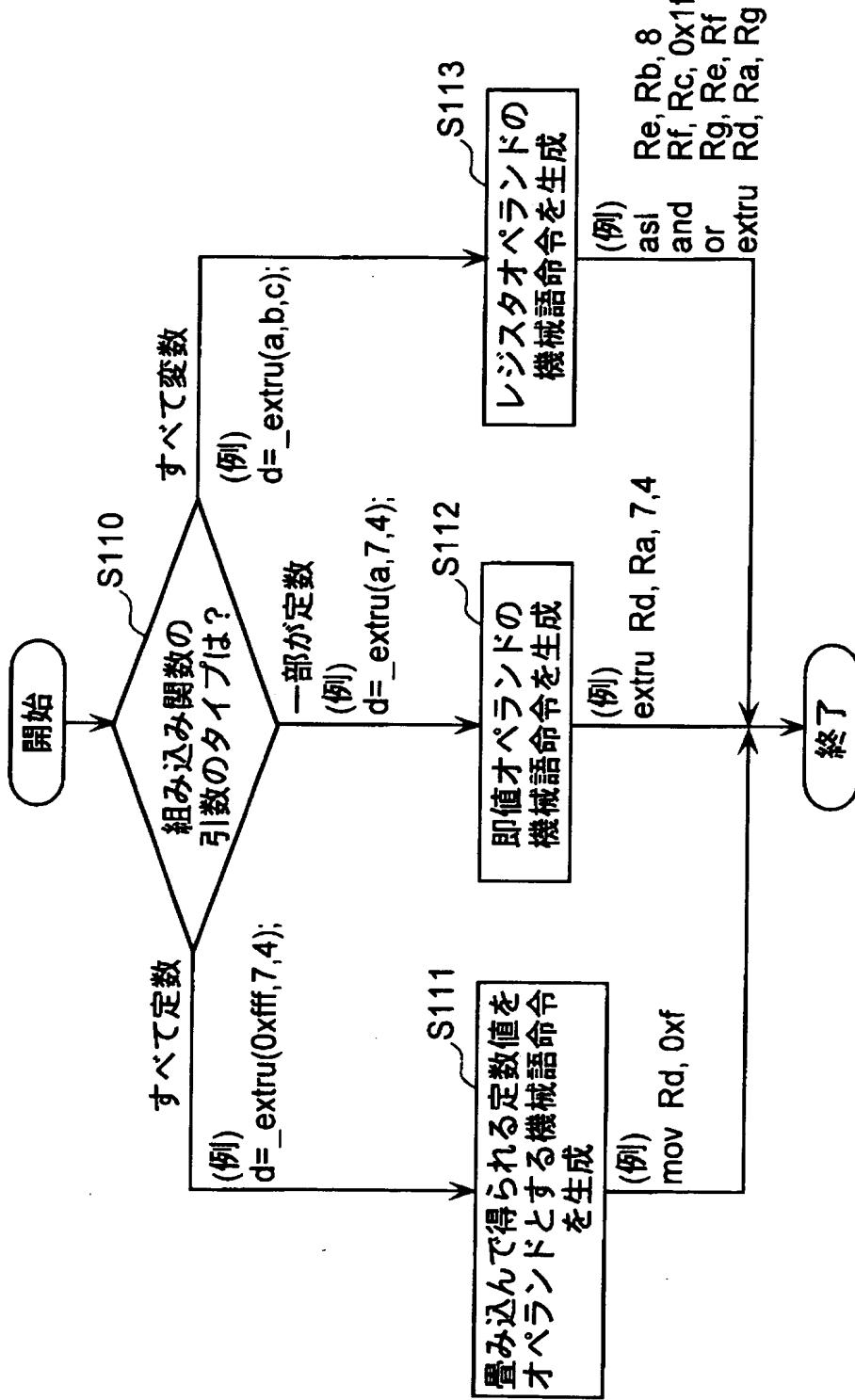
#endif /* __AMMPCC__ */
#endif /* __FUNCAMMP__ */

```

【図73】

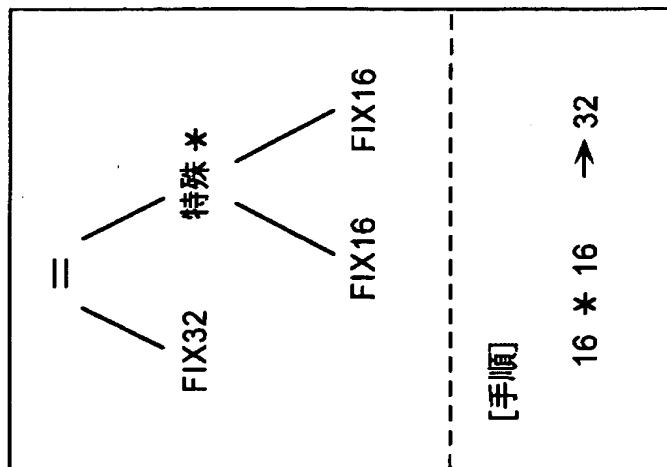


【図74】



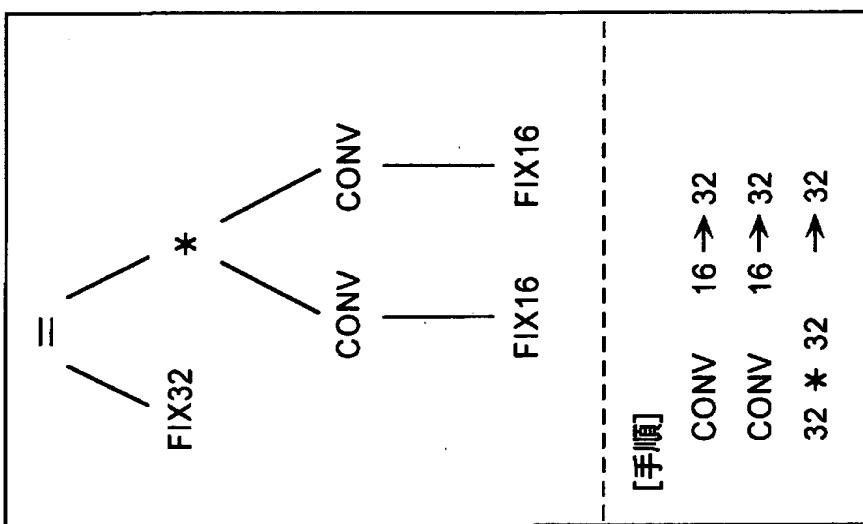
【図75】

(b) 最適化後の演算木



最適化

(a) 通常の演算木



【図76】

(a)

レイテンシ設定の例1

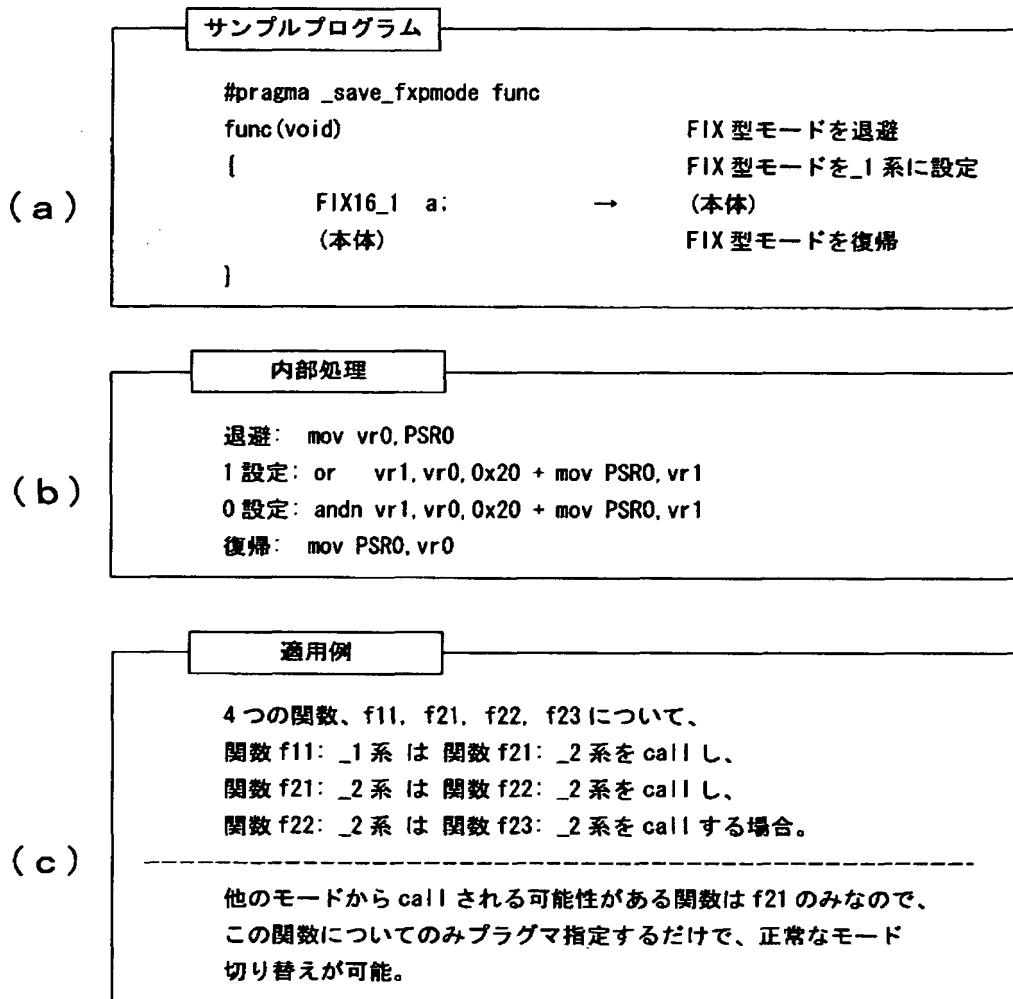
```
<例> wte と rde 間に 2 サイクル空ける
static inline int getbits(int a)
{
    int result ;
    asm (vr0 = a) {
        LATENCY L1, L2, 2 ;
        mov vr1, AVLD_BASEADDR ;
L1:     wte C0:C1, (vr1, AVLD_GETBITS), vr0 ;
L2:     rde C0:C1, vr2, (vr1, AVLD_READPORT) ;
    } (result = vr2) ;
    return result ;
}
```

(b)

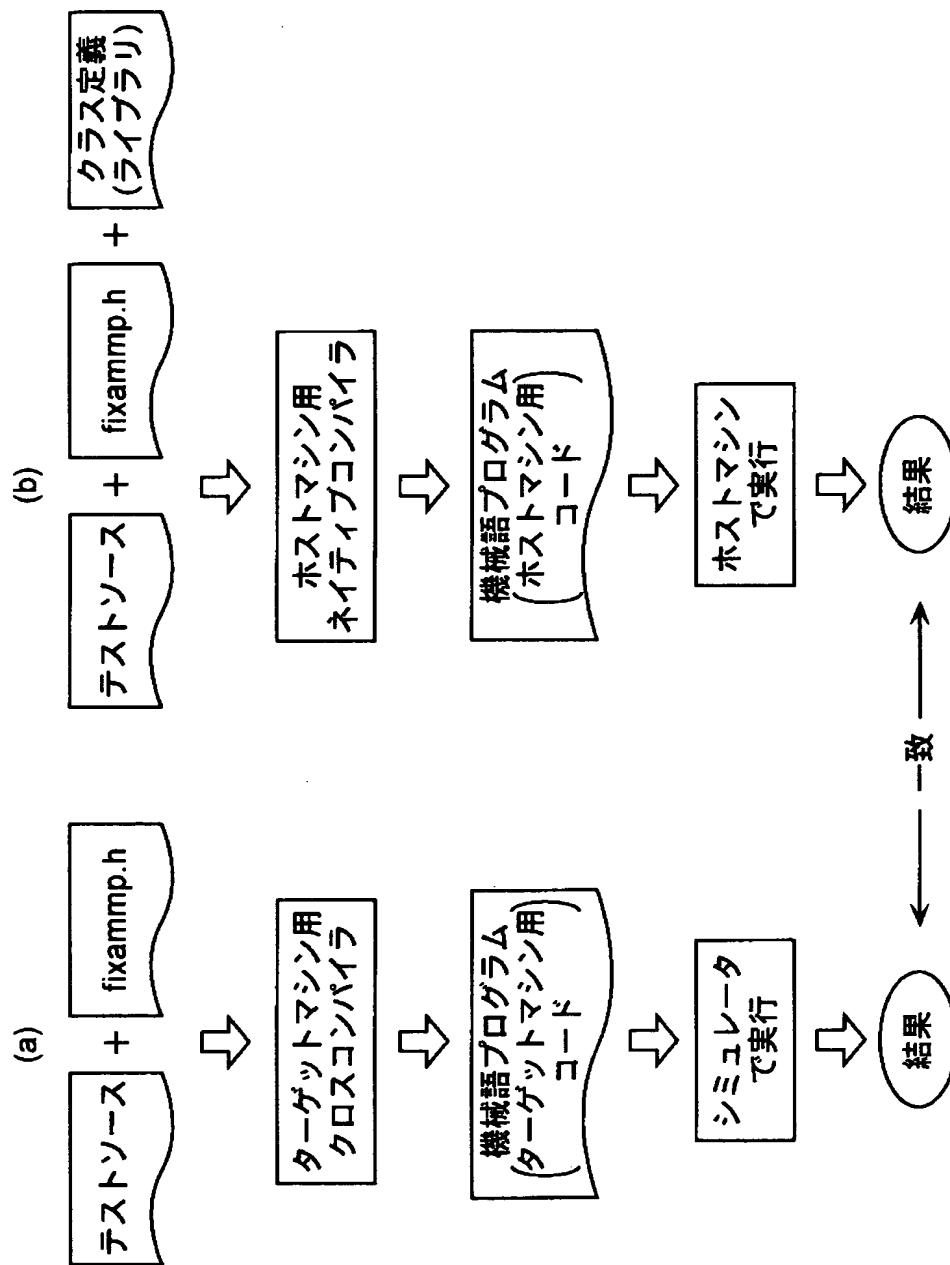
レイテンシ設定の例2

```
<例> wte 後 次の拡張レジスタアクセスまで 2 サイクル空ける
static inline void skipbits(int a)
{
    asm (vr0 = a) {
        mov vr1, AVLD_BASEADDR ;
        wte C0:C1, (vr1, AVLD_SKIPBITS), vr0, LATENCY(2) ;
    }
}
```

【図77】



【図78】



【書類名】 要約書

【要約】

【課題】 プロセッサが備える高機能な専用命令を効率的に生成することができ、かつ、コンパイラ自身のバージョンアップを頻繁に繰り返すことなく、機能拡張等の改良を行っていくことが可能なコンパイラを提供する。

【解決手段】 ソースプログラム101にインクルードされる演算子定義ファイル102等と、ソースプログラム101を機械語プログラム105に翻訳するコンパイラとから構成され、演算子定義ファイル102にはクラス定義による各種固定小数点型の演算子の定義が含まれ、コンパイラ100は、中間コードを生成する中間コード生成部121と、演算子定義ファイル102で定義されたクラスを参照している中間コードを機械語命令に置き換える機械語命令置換部122と、置き換えられた機械語命令を含む中間コードを対象として最適化を行う最適化部130等を備える。

【選択図】 図37

認定・付加情報

特許出願の番号	特願2002-226682
受付番号	50201151947
書類名	特許願
担当官	第七担当上席 0096
作成日	平成14年 8月 5日

<認定情報・付加情報>

【提出日】 平成14年 8月 2日

次頁無

出願人履歴情報

識別番号 [000005821]

1. 変更年月日 1990年 8月28日

[変更理由] 新規登録

住所 大阪府門真市大字門真1006番地  
氏名 松下電器産業株式会社